Grado en Ingeniería Informática

2018/2019

Trabajo Fin de Grado

Sistema de monitorización y alertas de estado para el laboratorio del Departamento de Informática

uc3m | Universidad Carlos III de Madrid

Escuela Politécnica Superior de Leganés

1 de julio de 2019

Autor: Aitor Alonso Núñez
Tutor: Alejandro Calderón Mateos

Agradecimientos

Deseo expresar mi agradecimiento al personal del laboratorio del Departamento de Informática: Óscar, Jaime y Roberto; quienes me han otorgado la posibilidad de realizar este proyecto poniendo a mi disposición la infraestructura y demás medios que he necesitado para su desarrollo. Asimismo agradezco como no puede ser de otro modo la buena acogida y buenas valoraciones que ha recibido el sistema por su parte.

Quiero manifestar también mi agradecimiento a los buenos docentes que he tenido a lo largo del grado, quienes han sabido ver en mí la persona curiosa e inquieta que soy y aprovecharlo para despertarme un interés y un amor por la ingeniería informática superiores a los que ya poseía cuando comencé estos estudios. En especial, quiero destacar y agradecer la labor de Alejandro Calderón Mateos, quien siempre ha tratado desinteresadamente de motivar, ayudar, enseñar, guiar y apoyar tanto a mis compañeros como a mí y cuya dedicación y amor por su oficio es envidiable a la par que contagiosa.

Por último pero no por ello menos importante, quiero agradecer a mi madre y a mi padre el constante esfuerzo que han realizado para que yo pueda cursar estos estudios, así como el continuo apoyo, reconocimiento, cariño y confianza que han depositado en mí, en quien soy, en quien quiero ser, en mis decisiones y en mis actos; como solo unos padres saben. A ellos especialmente, gracias.

Índice

1. English summary **1**

1.1. Introduction . . . 1

1.1.1. Motivation . . . 1

1.1.2. Goals and system scope . . . 2

1.1.3. Document structure . . . 3

1.2. About the final developed system . . . 4

1.2.1. Inheritance and system infrastructure . . . 4

1.2.2. System architecture . . . 5

1.2.2.1. Data center and servers monitoring subsystem . . . 5

1.2.2.2. Classrooms and computers monitoring subsystem . . . 6

1.3. User's satisfiability study . . . 7

1.4. Conclusions and future works . . . 7

1.4.1. Conclusions . . . 7

1.4.1.1. Related to the developed system . . . 7

1.4.1.2. Related to the development process . . . 8

1.4.1.3. Personal conclusions. Academic value . . . 8

1.4.2. Future works . . . 9

1.4.2.1. Google Calendar integration. . . . 9

1.4.2.2. Distributed architecture using Kubernetes . . . 10

1.4.2.3. Artificial intelligence to prevent situations on the data center . . . 10

1.4.2.4. Artificial intelligence to schedule maintenance tasks in classrooms . . . 10

2. Introducción **11**

2.1. Motivación . . . 11

2.2. Objetivos y alcance del proyecto . . . 12

2.3. Estructura del documento . . . 13

3. Estado de la cuestión **14**

3.1. Situación inicial e infraestructura heredada . . . 14

3.2. Estudio de alternativas y selección de base tecnológica . . . 14

3.2.1. Subsistema de monitorización del estado del CPD y servidores . . . 14

3.2.2. Subsistema de monitorización de las aulas y equipos . . . 16

3.3. Marco regulador . . . 20

3.3.1. Análisis de la legislación aplicable . . . 20

3.3.2. Estándares técnicos . . . 20

3.3.3. Cuestiones de propiedad intelectual . . . 20

3.4. Entorno socio-económico 22
3.4.1. Metodología utilizada y ciclo de vida 22
3.4.2. Planificación 24
3.4.3. Presupuesto 26
3.4.4. Impacto socio-económico 27

4. Análisis **29**
4.1. Requisitos de usuario 29
4.2. Casos de uso 34
4.3. Requisitos software 40
4.3.1. Requisitos funcionales 40
4.3.2. Requisitos no funcionales 45
4.4. Matriz de trazabilidad entre requisitos de usuario y requisitos de software 48

5. Diseño **51**
5.1. Definición de la arquitectura del sistema 51
5.1.1. Infraestructura heredada 52
5.1.2. Arquitectura de componentes de rpi1 53
5.1.3. Arquitectura de componentes de rpi2 54
5.1.4. Arquitectura de componentes de rpi3 55
5.2. Diseño de los scripts de monitorización 56
5.3. Diseños de las API REST 57
5.3.1. rpi2_api 57
5.3.2. rpi3_api 60
5.4. Diseños de las interfaces de usuario 63
5.4.1. Dashboard Grafana para el subsistema del CPD 63
5.4.2. Dashboard GatsbyJS para el subsistema de aulas 64

6. Implementación **67**

7. Implantación **68**
7.1. Preparación de la infraestructura del sistema 68
7.1.1. Dependencias necesarias en rpi1 68
7.1.2. Dependencias necesarias en rpi2 68
7.1.3. Dependencias necesarias en rpi3 69
7.2. Instalación y configuración de los componentes del sistema 69
7.2.1. Proceso de instalación 69
7.2.1.1. Proceso de instalación de rpi1 70
7.2.1.2. Proceso de instalación de rpi2 72

7.2.1.3. Proceso de instalación de rpi3 . . . 74
7.2.2. Configuración de los componentes . . . 76
7.2.3. Carga de datos . . . 77
7.3. Prueba de despliegue completa . . . 77
7.4. Plan de formación . . . 78
7.4.1. Plan de formación a usuarios . . . 78
7.4.2. Plan de formación de mantenimiento . . . 79

8. Evaluación **80**
8.1. Plan de pruebas . . . 80
8.1.1. Pruebas unitarias . . . 81
8.1.2. Pruebas de integración . . . 84
8.2. Estudio de la satisfacción de los usuarios . . . 87

9. Conclusiones y trabajos futuros **88**
9.1. Conclusiones . . . 88
9.1.1. Relativas al sistema desarrollado . . . 88
9.1.2. Relativas al proceso de desarrollo . . . 89
9.1.3. Personales. Valor y aporte académico . . . 89
9.2. Trabajos futuros . . . 90
9.2.1. Integración con Google Calendar . . . 90
9.2.2. Arquitectura distribuida utilizando Kubernetes . . . 91
9.2.3. Inteligencia artificial para adelantar acontecimientos en el CPD . . . 91
9.2.4. Inteligencia artificial para programar mantenimiento en las aulas . . . 91

10. Referencias **92**

11. APÉNDICE A. Acrónimos **95**

12. APÉNDICE B. Glosario **97**

13. APÉNDICE C. Galería del sistema **98**

Índice de figuras

1. Tablero visual Trello utilizado para gestionar el ciclo de vida del proyecto. Captura de pantalla. 23
2. Diagrama Gantt de la planificación del proyecto. Elaboración propia. 25
3. CU-01: Comprobar ocupación de las aulas para proceder a apertura de nuevos espacios. . . . 35
4. CU-02: Comprobar ocupación de las aulas para proceder a realizar mantenimiento no planificado. 36
5. CU-03: Comprobar temperatura y humedad del CPD. 37
6. CU-04: Situación de alarma en el CPD. 38
7. CU-05: Aviso de CPD en uso por el personal del Departamento. 39
8. Arquitectura del sistema. Elaboración propia. 51
9. Responsabilidades de la infraestructura que compone el sistema. Elaboración propia. 52
10. Arquitectura de componentes de la Raspberry Pi 1 (rpi1). Elaboración propia. 53
11. Arquitectura de componentes de la Raspberry Pi 2 (rpi2). Elaboración propia. 54
12. Arquitectura de componentes de la Raspberry Pi 3 (rpi3). Elaboración propia. 55
13. Jerarquía de los scripts de la Raspberry Pi 1 (rpi1). Elaboración propia. 56
14. Jerarquía de los ficheros de la Raspberry Pi 2 (rpi2). Elaboración propia. 57
15. Jerarquía de los ficheros de la Raspberry Pi 3 (rpi3). Elaboración propia. 60
16. Prototipo de la interfaz de usuario del subsistema del CPD. Panel de Grafana. Elaboración propia. 63
17. Prototipo de la interfaz de usuario del subsistema de aulas. Vista de reservas. Elaboración propia. 64
18. Prototipo de la interfaz de usuario del subsistema de aulas. Vista de aulas. Elaboración propia. 65
19. Prototipo de la interfaz de usuario del subsistema de aulas. Vista de error. Elaboración propia. 66
20. Repositorio git del proyecto en GitHub [28]. Captura de pantalla. 67
21. Estadísticas de lenguajes utilizados en el proyecto según GitHub. Captura de pantalla. 67
22. Estadísticas de lenguajes utilizados en el proyecto según cloc. Captura de pantalla. 67
23. El sistema revela como la climatización del ed. Torres Quevedo afecta a la climatización del CPD. 88
24. Rpi1 y módulos dentro del CPD. Temperatura actual en el panel led del módulo SenseHat. . . 98
25. Dashboard de Grafana completo. GUI del subsistema del CPD. 99
26. Monitor y bombilla de rpi2 sobre la puerta de acceso al CPD. Temperatura actual en el CPD. . 99
27. Monitor y bombilla de rpi2 sobre la puerta de acceso al CPD. Histórico de temperatura en el CPD. 100
28. Dashboard de GatsbyJS. Vista de reservas con reserva de prueba. GUI del subsistema del aulas. 101
29. Dashboard de GatsbyJS. Vista de aulas (4.0.F18). GUI del subsistema del aulas. 101
30. Dashboard de GatsbyJS. Vista de aulas (2.2.C05). GUI del subsistema del aulas. 102
31. Dashboard de GatsbyJS. Error web caída. Vista de reservas. GUI del subsistema del aulas. . 102

Índice de tablas

1. Comparativa alternativas dashboards. . . . 15
2. Comparativa alternativas lenguajes de programación para desarrollar los scripts. . . . 16
3. Comparativa alternativas lenguajes de programación para desarrollar la/s API REST. . . . 17
4. Comparativa alternativas frameworks JavaScript para desarrollar el dashboard. . . . 18
5. Comparativa alternativas generadores de contenido estático para React. . . . 19
6. Presupuesto: coste de personal con cargo al proyecto. . . . 26
7. Presupuesto: coste de software con cargo al proyecto. . . . 26
8. Presupuesto: coste de hardware con cargo al proyecto. . . . 26
9. Presupuesto: coste final del proyecto. . . . 27
10. Plantilla requisitos de usuario. . . . 29
11. RU-01: Visualización de la temperatura dentro del CPD. . . . 30
12. RU-02: Visualización de la humedad dentro del CPD. . . . 30
13. RU-03: Visualización del nivel de carga del SAI. . . . 30
14. RU-04: Visualización del estado de la iluminación del CPD. . . . 30
15. RU-05: Alerta de problemas y situaciones anómalas en el CPD. . . . 30
16. RU-06: Integración de las métricas del CPD con Zabbix. . . . 31
17. RU-07: Visualización del estado de las aulas. . . . 31
18. RU-08: Visualización de la ocupación (número de usuarios) de las aulas. . . . 31
19. RU-09: Visualización de la planificación diaria (reservas) de las aulas. . . . 31
20. RU-10: Visualización del estado de los equipos de las aulas. . . . 31
21. RU-11: Disposición de los equipos de las aulas en la interfaz. . . . 32
22. RU-12: Utilizar un monitor externo para mostrar las métricas del CPD. . . . 32
23. RU-13: Utilizar un monitor externo adicional para mostrar la información de las aulas. . . . 32
24. RU-14: Estilo de la GUI del subsistema de aulas. . . . 32
25. RU-15: Resumen de la información del CPD en el monitor del subsistema de aulas. . . . 33
26. RU-16: No invasivo con la infraestructura actual. . . . 33
27. RU-17: Creación de script de instalación. . . . 33
28. Plantilla casos de uso. . . . 34
29. CU-01: Comprobar ocupación de las aulas para proceder a apertura de nuevos espacios. . . . 35
30. CU-02: Comprobar ocupación de las aulas para proceder a realizar mantenimiento no planificado. 36
31. CU-03: Comprobar temperatura y humedad del CPD. . . . 37
32. CU-04: Situación de alarma en el CPD. . . . 38
33. CU-05: Aviso de CPD en uso por el personal del Departamento. . . . 39
34. Plantilla requisitos software. . . . 40
35. RF-01: Monitorización de la temperatura actual del CPD. . . . 40

36. RF-02: Visualización de la temperatura actual del CPD en la interfaz del sistema. 41
37. RF-03: Monitorización de la humedad actual del CPD. 41
38. RF-04: Visualización de la humedad actual del CPD en la interfaz del sistema. 41
39. RF-05: Obtención de los datos de nivel de carga del SAI. 41
40. RF-06: Visualización del nivel de carga actual del SAI en la interfaz del sistema. 41
41. RF-07: Monitorización del estado actual de la iluminación del CPD. 41
42. RF-08: Visualización del estado actual de la iluminación del CPD. 42
43. RF-09: Alerta visual de temperatura elevada en el CPD. 42
44. RF-10: Alerta sonora de temperatura elevada en el CPD. 42
45. RF-11: Alerta visual de desconexión de la red eléctrica del rack del LDI. 42
46. RF-12: Alerta sonora de desconexión de la red eléctrica del rack del LDI. 42
47. RF-13: Obtención de los datos de reservas de las aulas. 43
48. RF-14: Diseño de la interfaz de usuario del subsistema de aulas. 43
49. RF-15: Resumen de la información del CPD en la interfaz del subsistema de aulas. 43
50. RF-16: Visualización del estado de las aulas. 43
51. RF-17: Visualización de la ocupación (número de usuarios) de las aulas. 44
52. RF-18: Visualización de la planificación diaria (reservas) de las aulas. 44
53. RF-19: Visualización del estado de los equipos de las aulas. 44
54. RF-20: Disposición de los equipos de las aulas en la interfaz. 44
55. RF-21: Identificación del aula que se muestra en el panel principal. 45
56. RNF-01: Utilización del módulo de sensores SenseHat para las métricas del CPD. 45
57. RNF-02: Envío de los datos de temperatura y humedad del CPD a Zabbix. 45
58. RNF-03: Visualización de la temperatura actual del CPD en el panel led del módulo SenseHat. 45
59. RNF-04: Utilización del módulo de cámara para monitorizar la iluminación del CPD. 46
60. RNF-05: Utilización de un monitor para la interfaz del subsistema del CPD. 46
61. RNF-06: Utilización de un monitor para la interfaz del subsistema de aulas. 46
62. RNF-07: Comunicación entre subsistemas mediante API REST. 46
63. RNF-08: Autenticación en las peticiones de escritura a las API. 47
64. RNF-09: Protección por cortafuegos de las API. 47
65. RNF-10: Protección por cortafuegos de las interfaces de usuario. 47
66. RNF-11: Compatibilidad con la arquitectura de procesador ARM. 47
67. RNF-12: Optimización del sistema. 47
68. RNF-13: Ahorro de energía de los monitores. 48
69. RNF-14: Creación de script de instalación. 48
70. RNF-15: Protección ante fallos de red y pérdidas de conexión. 48
71. Matriz de trazabilidad entre requisitos de usuario y requisitos de software. 50
72. Respuesta petición GET / rpi2_api. 58

73. Respuesta petición GET /cpd-status rpi2_api. 58
74. Formato petición POST /cpd-status rpi2_api. 58
75. Respuesta petición GET /cpd-status rpi2_api. 58
76. Respuesta error método no permitido rpi2_api. 59
77. Respuesta error de autenticación 1 rpi2_api. 59
78. Respuesta error de autenticación rpi2_api. 59
79. Respuesta error mal formato del cuerpo de la petición rpi2_api. 59
80. Respuesta errores internos rpi2_api. 59
81. Respuesta GET / rpi3_api. 61
82. Respuesta GET /reservations rpi3_api. 61
83. Respuesta GET /classrooms rpi3_api. 61
84. Respuesta GET /occupation rpi3_api. 62
85. Respuesta error método no permitido rpi3_api. 62
86. Respuesta errores internos rpi3_api. 63
87. Sintaxis y valores por defecto fichero de configuración JSON. 77
88. Comprobaciones prueba de despliegue completa. 78
89. Plantilla pruebas. 80
90. PU-01: Obtención del estado del CPD (rpi2_api). 81
91. PU-02: Obtención de la temperatura del CPD mediante parámetros en la URL (rpi2_api). . . . 81
92. PU-03: Obtención de la temperatura y humedad del CPD filtrando dos parámetros en la URL (rpi2_api). 82
93. PU-04: Actualización de la temperatura del CPD (rpi2_api). 82
94. PU-05: Petición con método no permitido (rpi2_api). 82
95. PU-06: Petición POST sin cabecera de autenticación (rpi2_api). 83
96. PU-07: Petición POST con cabecera de autenticación vacía (rpi2_api). 83
97. PU-08: Petición POST con cabecera de autenticación con token erróneo (rpi2_api). 83
98. PU-09: Mal formato del cuerpo de la petición, llave de cierre (rpi2_api). 84
99. PU-10: Simulacro de alarma temperatura elevada en el CPD. 84
100. PI-01: Arranque automático del servicio de temperatura del CPD. 85
101. PI-02: Monitorización de la iluminación del CPD. Comunicación entre componentes. 85
102. PI-03: Simulacro de alarma corte eléctrico en el rack del LDI. 85
103. PI-04: Carga externa del dashboard GatsbyJS. 86
104. PI-05: Reflejo de las actualizaciones de reservas del sistema externo. 86
105. PI-06: Solicitud a instalador del estado de los equipos. 86

English summary

Introduction

Motivation

One of the tasks the personnel at the Laboratory of the Computer Science and Engineering Department (LDI from Spanish) do is to check and maintain the physical infrastructure of the Department, such some servers racks and computers classrooms. For that, the staff have to complete a series of **daily tasks**, which involve the need for quick access to some information about the Laboratory services status.

One of these tasks is to guarantee that the **temperature** inside the data center (the room where the servers are) is on an **appropriate range**. For that, staff have to daily check a thermometer inside the room and adjust the refrigeration accordingly.

Another of these tasks is to prepare and guarantee normal operation and performance in **computer classrooms**, which are used for lessons taught by the Computer Science and Engineering Department. These classrooms are also **used by students** to do their academic work in their free time because the LDI environment is where their works will be evaluated later.

For that, the LDI owns two classrooms on the Torres Quevedo building (hall 4.0.F) and on the Sabatini building (hall 2.2.C). Normally the classrooms on the Torres Quevedo building are open so then **students can use it**. However, **when these classrooms are occupied students come to the LDI office to ask for opening a classroom** on the Sabatini building. On that moment, LDI personnel have **to check the current and future status** of each classroom before to decide if they should open a new classroom or not. For that, they have to check some registers, logs files and data distributed across multiple web services. Sometimes even it is necessary to run a remote command on a control server.

The LDI also has a server (called guernika) to which students can connect and do their academic work with the same environment which is in the classrooms, but **most of the students prefer to do their job in a dedicated computer** instead of share resources with their classmates.

Those **tasks are mostly monotonous** (monitoring the servers and data center temperature, to plan the daily tasks depending on the occupation of the classrooms, to monitor the status of the computers on the classrooms, etc.) **and easy to automate**. With the automation of these tasks,

LDI staff would save time on monitoring and maintenance routine tasks. A time that they could invest in upgrading the current infrastructure. Also, automatic control and continuous monitoring will allow LDI personnel **to anticipate unexpected events or system failures**, reducing their adverse effects or even avoiding them.

On the situation explained above, I expose the possibility of automating the control and monitoring of the classrooms and part of the servers and the data center itself. For that, I suggest the development and implementation of a **status alerts system**. This system will **monitor the data center** status and inform of strange or dangerous events on it or in the LDI own rack and servers. It will also **monitor the status of the classrooms**: their reservations, their occupation and the status of the computers on them; aiming to simplify this information access.

An **in-depth analysis** of the needs of the LDI itself and the personnel had been needed to design and implement the future system. The goal of this project has always been **to develop a useful and beneficial system to the LDI services, which must be simple and easy to use, understand and maintain**. On this in-depth analysis it has been necessary to identify the core ideas of the future system to set the system goals, trying to avoid distractions that could complicate the system or make it superfluous. Therefore it has been necessary a lot of work in the shadows that is **difficult to capture** on this document.

After this in-depth analysis of the LDI service needs and sharing ideas with the LDI staff, I got approval for the system development with the following **list of goals** the system should accomplish.

Goals and system scope

The system goals can be summarized as the following:

- **To monitor the data center** and server status: temperature, humidity, room light (switched on/off) and power supply.
- **To monitor the classrooms status** and the status of their computers: classroom reservations, classroom occupation, computer occupation, and computer status.
- To present all the system information in a way that should be **understandable and comprehensible at a glance**.
- **To alert the LDI staff** of every unusual or strange situation and failures on the data center or the classrooms as soon as possible.

The planned system scope is inside the LDI environment, in particular, the LDI office (4.0.F11).

The alerts will be fired physically on the LDI office (audiovisual and loud alarms) and/or by email. The idea is to put some **informative panels or monitors on the walls inside the office** that allow the LDI staff to get the information they need at a glance, and also to allow the rest of the Computer Science and Engineering Department people who come to the data center to check its conditions.

LDI personnel will obtain this information, in addition from those panels or monitors, from their own work computers as usual, but in a **centralized way on a quick and simple service**, with the intention to make their work easier.

Document structure

This document is divided into nine chapters with the following structure.

The first and current chapter, English summary, contains an English synopsis of the Spanish document. This synopsis includes the document introduction, the project goals, an explanation about the final developed system, the user's satisfiability study and its conclusions.

The second chapter, Introducción, include the motivation, goals, scope of the project, and the document structure.

The third chapter, Estado de la cuestión, analyzes the starting situation I faced to when I decided to work on this project. It also contains the study of alternatives and the technological base selection. In this chapter, we can find the **regulatory framework** of the project, the **socio-economic environment** of it, and also a summary of the development methodology used on the project.

The fourth chapter, Análisis, includes all the aspects related to the analysis phase of the project. Here we can find the requirements of the system (user and software requirements) and some use cases.

The fifth chapter, Diseño, documents the system architecture, interfaces, and mockups defined for the system. It contains some diagrams and figures which help to understand the system, especially its architecture and the relation between its components and the inheritance infrastructure.

The sixth chapter, Implementación, contains some information about the source code and the rest of the implemented software components, which its implementation has been necessary to develop the system.

The seventh chapter, Implantación, documents the system deployment process, technical needs, and the software dependencies. It details also the formation plans.

The eighth chapter, Evaluación, establish the test plan carried out to assure the quality and the correct functioning of the final system. This chapter also presents the user's satisfiability study.

The ninth and last chapter, Conclusiones y trabajos futuros, includes a series of technical and personal conclusions about the developed system and future works I plan to do on the system to improve its functionality.

About the final developed system

The system has been developed following an **agile software development methodology** which is Kanban methodology. This methodology allows an easy and at a glance organization of the project tasks by using a Kanban board and **continuous delivery** workflow.

Inheritance and system infrastructure

Before starting the project, a **previous infrastructure** existed which is intended to be used as a part of the new system. Three current servers must be highlighted: ultraheroe, instalador and web. **Ultraheroe** will hold some components such as Zabbix and Grafana dashboard used to store and show data center metrics respectively. **Instalador** will allow the system to check the current status of the computers in the classrooms and classrooms occupation. Finally, **web** will be from where we will get the information about daily classrooms reservations.

LDI provides the following **components to implant the system** which will be used for the project. These components are three **Raspberry Pi 3 model B+** computers, a SenseHat Raspberry Pi sensors module, a camera Raspberry Pi module, two computer monitors and a Wi-Fi Philips®Hue White and Color Ambiance light bulb. These Raspberry Pi computers have the following specifications:

- ARM 64 bits quad-core processor at 1.4GHz.
- 1GB LPDDR2 SDRAM main memory.
- Micro SD as the system disk.
- Network connectivity: IEEE 802.11 b/g/n/ac and Gigabit Ethernet.
- HDMI audio/video output.
- 5V at 2.5A power supply.

System architecture

The system is divided into **two main subsystems**: the **data center** and servers monitoring subsystem and the **classrooms** and computers monitoring subsystem.

Data center and servers monitoring subsystem

The **data center** monitoring subsystem is composed of **two Raspberry Pi computers** (called **rpi1** and **rpi2**), one monitor, the SenseHat and camera modules and the Wi-Fi light bulb.

Rpi1 is inside the data center room and has attached the SenseHat and camera modules, which are controlled by some **Python 3 scripts**. They use the SenseHat module to measure temperature and humidity in the room every 60 seconds. The values read are copied to `/tmp` as text files. A **Zabbix** agent which is running on rpi1 reads these values from the files and sent them to Zabbix on **ultraheroe**. The camera module is used to take a picture of one wall in the room every 10 seconds and then analyze the brightness of the photo to determine if the light inside the room is switched on or off.

Rpi2 is located in the LDI office over the door that gives access to the data center. It's plugged to a monitor that displays a **Grafana** instance running on ultraheroe that **shows the data center metrics** read by rpi1. This Grafana instance gets the data directly from Zabbix. Grafana shows the current temperature and humidity values, the last week temperature historical graph, and the LDI rack UPS current battery level.

There is an **API REST** running on rpi2 which is written in Go, called **rpi2_api**. This API receives the temperature, humidity and light status data on HTTP requests from rpi1. These request are authenticated by a Bearer token. Rpi2_api gets the UPS status in the same way but from ultraheroe server. All data on the API can be obtained by HTTP GET requests. This API controls the **Wi-Fi light bulb**, which is placed next to the monitor.

The Wi-Fi light bulb copies the **state of the light inside the data center** room. If the light inside the room is switched on, then the light bulb will be on. It will be off otherwise. This allows LDI staff to know if there is some non-LDI people working inside the data center, so then they know if they can leave and close the office to do some other task on the classrooms.

If rpi2_api detects that the data it receives contains dangerous values or implies some dangerous state (temperature is high, UPS battery level is getting lower) it **fires an alarm on the office**. The Wi-Fi light bulb starts blinking in red and a siren sound is played on the monitor

speakers. The light bulb alarm blinking has priority over whatever another light bulb state.

Classrooms and computers monitoring subsystem

The **classrooms** monitoring subsystem is composed of one Raspberry Pi (called rpi3) and one monitor. The monitor is placed on a wall inside the LDI office, between the two doors that give access to it.

There is another**API REST** running on rpi3, called rpi3_api. This API is also written in Go and it is responsible for getting the classrooms reservations and computer status. Analogous to rpi2_api, all information on this API can be obtained with HTTP GET requests.

Rpi3 is also running a **self-development server** written in Go, which is serving a **self-development dashboard** coded in JavaScript using the GatsbyJS framework for react. This dashboard displays the classrooms reservations for today, classroom occupation (number of users per classroom and the computers being used) and computers status (if they are powered off, running GNU/Linux, running Windows or in some failure state).

The **reservations** for the current day are obtained from the LDI **web** using a **web crawler** implemented on the API. This is due to a user requirement about the **new system being as minimally invasive as possible** with the inheritance infrastructure. Computers status and occupation are gotten from **instalador** server by running some **yet implemented python scripts** whose output need to be parsed by our system. The output of the scripts is human-readable, so the implementation of a **parser** in rpi3_api was needed.

The GatsbyJS dashboard asks rpi3_api for all this data and prints it on the main panel defined in the dashboard design. This **main panel** is **rotative** and shows different views: the next eight **reservations for today** (divided into two views with four reservations each) and a view for each **classroom**. Classrooms views display the computers in them following the same layout they have on the classroom. For each computer is shown its current state and whether someone is using it.

This dashboard also has a permanent **aside panel** which shows the current classroom reservation (occupied, free, near to reservation) and the users on each, as a summary. The rest of the screen is filled with a **header** containing the LDI name and a clock, and a **footer** with the data center information in a summarized way.

User's satisfiability study

After a few weeks from the system implantation, I did a small **interview** with the LDI personnel to check the users' **satisfiability** about the product.

LDI staff have shown to be satisfied with the developed system. The system meets all the initial users' requirements. Its implantation has saved their time during daily tasks and has also helped LDI personnel to better scheduling some maintenance tasks. Also, **the system has allowed detecting some situations that must be corrected which were ignored until today**. These situations are explained in more detail in the following section.

Conclusions and future works

Conclusions

Related to the developed system

The client is satisfied with the system and its implantation has allowed detecting some unknown situations.

- Every Monday in the morning, around 10:00 and 11:00, the LDI rack loses power supply and UPS comes into battery mode. This is a **brief cutoff** of around 1-2 seconds. LDI personnel is investigating the main cause, but it is thought that this is due to a weekly test that the UPS itself does.
- The **climatization of the Torres Quevedo building affects** notoriously to the temperature inside the data center. The problem seems to be some leak in the facility, due to the fact that the climatization system of the building is supposed to be deactivated on the data center room (4.0.F09). LDI staff has informed the maintenance team and they are currently inspecting the building climatization system.

On figure 23 an example of the data recorded during February is shown, where it can be appreciated how the **heating of the building is affecting the temperature inside the data center room**. Nevertheless this room is supposed to be isolated from the climatization system of the building and has its own cooling machines. On the figure, it is shown the data recorded from Monday 11th to Friday 15th February 2019. The oscillation on the temperature corresponds with the opening hours at the university.

Also, **the developed system accomplishes the goals defined for the project**. Therefore, it is concluded that the system fits the purpose for what it was originally thought. Also, from the user's satisfiability study it can be concluded that the system implementation results in a **useful and favorable acquisition for the LDI environment and its daily work**.

An appendix with some graphical user interface screenshots and pictures of the final developed and implemented system is attached at the end of the document (APÉNDICE C. Galería del sistema).

Related to the development process

The development process **met the planning period** since the different phases had begun and finished on the originally planned dates. Also, the usage of an agile software development methodology with a feature such as the continuous delivery has promoted the product's improvement **to correctly supply the client needs**.

For these reasons, it can be concluded that **the development process had been positive** since its out of incidents completion. Also, the decision to use an **agile software development methodology** such Kanban is considered **to be a wise choice**. The use of this methodology on the office where the product would be implemented has allowed making a **fit and personalized product** to the client's needs and its current infrastructure.

Personal conclusions. Academic value

The **knowledge** I have **acquired** during my computer science and engineering degree studies has allowed me to make this project possible. Its completion makes me especially proud because of the good reception from my workmates at the LDI. Likewise I am honored to be able to appreciate the system's utility during my last days working at the LDI and to acknowledge how it has improved and simplified the daily work.

This project would not be possible without the knowledge and skills acquired on my degree subjects, especially:

- **Programming.** From this subject I learned to write and understand source code, which has allowed me to acquire on my own Python, Go and JavaScript knowledge that the system has required.
- **Computer networks.** The knowledge about computer networks and the protocol stack, especially its Internet and application layers, have allowed me to develop an effective and efficient communication channel between the different components of the system.
- **Software engineering.** This subject showed me the importance of clean and maintainable code, as well as its separation on well-defined functions and files. I have carried these principles as standard during the system implementation.

- **Software development projects management.** Without the knowledge and the experience acquired on this subject, I would be able to correctly plan the system life cycle. Also, the skills acquired on this subject have allowed me to lean by myself an agile software development methodology such as Kanban and apply it to this project.
- **User interfaces.** Here is where I have learned the importance of good practices, heuristics and user's cognitive process to design simple and intuitive graphical user interfaces, which was one of the system goals.
- **Operating systems.** From this subject, I got part of my Bash and GNU/Linux system architecture knowledge which I have needed to the implantation and deployment of the system, as well to work with the Raspberry's hardware components.
- **Cryptography and computer security.** On this subject, I acquired knowledge about authentication and signature processes needed to ensure the communication between the different components, as well as how to use the public key cryptography systems (such RSA) used on this project.

Personally, I can conclude that **I have acquired the knowledge and skills** I aimed when four years ago I applied to be a student of the computer science and engineering degree at this university. I am proud to have demonstrated it with this project, which also results in a **useful tool** for a service such as the LDI, which its labor is indispensable for the correct operation and quality of my degree studies as I could appreciate while working with them.

Future works

Next, I present some improvements and future works that could be applied to the system. **I plan to implement most of them** during the days I remain at the LDI. However, I would like to remind that the work documented in this report has been fitted to the work hours related to a final degree project, which correspond with 360 hours (12 ECTS x 30 hours each) so I have not had the material time to address them inside my final degree project.

Google Calendar integration.

In the LDI we use **Google Calendar** and the university Google accounts to schedule and organize some maintenance tasks, meetings, and other activities that require the presence of multiple personnel. The classroom subsystem is a good choice to show in its main panel a view with the **remaining task for the current day** or even the current week.

Distributed architecture using Kubernetes

Currently the system divides its components between the three Raspberry Pi computers. This implies that the system has less reliability about hardware failures than it would if it were running on a single computer. The failure of any of the three computers may affect the rest of the system, therefore there are more **potential points of failure**.

Given the system architecture and its magnitude, it will be acceptable to migrate it from running on the Raspbian over the dedicated Raspberry Pi computers to **Kubernetes** pods. Using Kubernetes, whose implementation for this project should be direct and natural, would give the system an **additional compatibility layer**. As a result, all the system environment being inside Docker containers will allow to replicate and launch the system on whatever other hardware infrastructure, not necessarily the Raspberry Pi computers.

Also, the use of Kubernetes would give the system a **better hardware failure reliability**. That is because if a pot or node gets down for whatever reason it can be relaunched on another hardware. This also allows making full maintenance or reinstallation of the Raspberry Pi computers without the need for temporary stop the system.

Artificial intelligence to prevent situations on the data center

The control and monitoring of the data center environment is a very important responsibility at the LDI. Adding artificial intelligence techniques, especially **machine learning** techniques, will allow the system to identify some repetitive or common situations on the data center as well as simulated situations that could be used to train the system. With this knowledge, it could be possible **to prevent** some situations in the data center **and inform the LDI personnel** about those even **before** they occur.

Artificial intelligence to schedule maintenance tasks in classrooms

LDI personnel have to ensure the correct operation of the classroom's equipment, for what some maintenance tasks are needed, althought these must not interfere with the normal workflow and utilization of the classrooms. With the addition of **artificial intelligence** techniques, the system could **recommend** which periods are more suitable for maintenance tasks, or which computers are under bigger wear and must be inspected with special attention. Also, if the system is trained on historical data it could **predict** when a computer or a hardware component will fail based on a probability or time window.

Introducción

Motivación

Entre las labores de las que se encarga el personal del Laboratorio del Departamento de Informática (LDI) se encuentra el control y mantenimiento físico de infraestructura del Departamento, como son ciertos racks de servidores y aulas docentes de informática. Con este objetivo, los técnicos del LDI realizan diariamente una serie de **tareas fijas** en las que habitualmente requieren de acceso rápido a alguna información para tener conocimiento del estado actual del servicio.

Una de estas tareas es la de asegurar que la **temperatura** dentro del CPD (sala en la que se encuentran los servidores) se mantenga en un **rango adecuado**. Para ello, diariamente se revisa un termómetro dentro de la sala y se ajusta la climatización en la misma.

Otra de estas tareas es la de preparar y asegurar el funcionamiento de las **aulas docentes informáticas** para las asignaturas impartidas por el Departamento, así como asegurar en la medida de lo posible que **los estudiantes disponen de un entorno** para desarrollar sus prácticas idéntico al que utilizan cuando reciben clase y en el que se les evalúa.

El LDI dispone de dos aulas en el edificio Torres Quevedo (pasillo 4.0.F) y en el edificio Sabatini (pasillo 2.2.C). Habitualmente se encuentran **abiertas y a disposición de los estudiantes las aulas del edificio Torres Quevedo**, ya que además el despacho principal del LDI se sitúa enfrente (4.0.F11). Sin embargo, **cuando estas aulas se encuentran ocupadas, los estudiantes acuden al despacho del LDI para solicitar la apertura** de un aula del edificio Sabatini. En esos momentos, los técnicos deben **revisar el estado actual y futuro** de todas las aulas antes de proceder o no a la apertura. Para ello hay que revisar varios registros, datos separados en diversas webs y ficheros de log. En ocasiones incluso se requiere la ejecución de algunos comandos desde un servidor que se utiliza para gestinar las aulas y lanzar tareas en las mismas.

Si bien es cierto que el LDI dispone de un servidor (*guernika*) que pueden utilizar los estudiantes para sus prácticas y que es exactamente el mismo entorno que las aulas, **muchos estudiantes prefieren trabajar directamente en las aulas** en un equipo dedicado, en lugar de trabajar en remoto en el servidor compartiendo recursos con el resto de sus compañeros.

Estas **tareas** son en su mayoría **monótonas** (monitorizar la temperatura de los servidores en el CPD, planificar el día conforme a la ocupación de las aulas, monitorizar el estado de los equipos en las aulas, etc.) **y fácilmente automatizables**. Con la automatización de estas tareas, **los técnicos del LDI ahorrarían tiempo** en tareas rutinarias de control y mantenimiento que podrían invertir en mejorar la infraestructura actual. Asimismo, un control automático y una monitorización continua permitiría a los técnicos **adelantarse a situaciones imprevistas o fallos** en los sistemas y la infraestructura, impidiéndolos o mitigando enormemente los posibles efectos adversos.

Así, planteo la posibilidad de automatizar el control y monitorización de las aulas y parte del CPD. Para ello sugiero el desarrollo e implementación de un **sistema de alertas de estado** que por una parte **monitorice el CPD** y alerte de posibles situaciones anómalas en el mismo y en el rack de servidores del LDI y que por otra parte **monitorice el estado de las aulas**, su ocupación, reservas y el estado de los equipos en las mismas; simplificando el acceso a esta información.

Para concebir el futuro sistema ha sido preciso un **análisis en profundidad** de las necesidades del servicio y los técnicos. La intención de este proyecto siempre ha sido la de desarrollar un **sistema útil y beneficioso para el servicio, a la par que simple y fácil de utilizar y mantener**. Para ello, se ha requerido como digo un análisis en profundidad donde se ha necesitado identificar y destacar lo importante para la definición de los objetivos del sistema, cuidando de no caer en distracciones que pudieran complicar este ni derivar en un sistema superfluo. Ha sido necesario, pues, mucho *trabajo en la sombra* que es **difícil plasmar** en el presente documento.

Tras el análisis en profundidad de las necesidades del servicio y la puesta en común de ideas con el personal del LDI, recibo el visto bueno al desarrollo del sistema y se concluye siguiente la **lista de objetivos** que este deberá cubrir.

Objetivos y alcance del proyecto

Los objetivos del sistema a desarrollar se pueden resumir en los siguientes:

- **Monitorizar el estado del CPD** y los servidores: temperatura, humedad, iluminación interior (encendida/apagada), y estado de la red eléctrica.
- **Monitorizar el estado de las aulas** y los equipos en las mismas: reservas de aulas, ocupación de las aulas y los equipos, y estado de los equipos.
- Presentar toda la información de forma que sea **asimilable y comprensible de un vistazo**.
- **Alertar a los técnicos** de situaciones extrañas, anómalas o malfuncionamientos bien en el CPD o en las aulas tan pronto como se detecten.

Así pues, el alcance previsto del sistema es dentro del entorno del LDI, concretamente en el despacho 4.0.F11. Las alertas se ejecutarían físicamente en el despacho (audiovisuales y sonoras) y/o por correo electrónico. La idea es colocar un par de paneles y **pantallas informativas en las paredes** dentro del despacho que permitan a los técnicos obtener la información que necesitan de un vistazo, así como al PDI que accede al CPD comprobar las condiciones del mismo.

Los técnicos podrán obtener la información, además de desde los paneles, desde sus propios equipos de trabajo como hasta ahora, pero de manera **centralizada en un único servicio y de forma rápida y sencilla**. Con este objetivo se espera facilitar el acceso a esta información y en consecuencia las tareas diarias.

Estructura del documento

El presente documento está dividido en nueve capítulos como se puede apreciar en el índice del mismo.

El primer capítulo, English summary, se corresponde con un resumen en inglés del presente documento tal y como se exige en la normativa de mi titulación para los TFG (plan 2011).

El segundo y actual capítulo, Introducción, comprende la motivación, objetivos y alcance del proyecto; así como esta explicación sobre las secciones del documento.

El tercer capítulo, Estado de la cuestión, analiza la situación inicial que me encontré a la hora de decidir desarrollar este proyecto, así como el estudio de alternativas y la selección de base tecnológica para el desarrollo. En este capítulo se encuentran también recogidos el **marco regulador** que afecta al proyecto y el **entorno socio-económico** del mismo, así como una introducción al ciclo de vida del producto y la metodología utilizada para el desarrollo del sistema.

El cuarto capítulo, Análisis, comprende todos los aspectos relativos a la fase análisis del sistema. Así pues, este capítulo comprende la identificación de requisitos (de usuario y software) del sistema y una presentación de diversos casos de uso.

El quinto capítulo, Diseño, documenta la arquitectura, interfaces y prototipos definidos en el sistema. En este capítulo se incluyen algunos diagramas y figuras para ayudar a comprender mejor el sistema y en concreto la arquitectura de este y la relación entre sus componentes y la infraestructura heredada.

El sexto capítulo, Implementación, recoge cierta información sobre el código fuente y demás componentes software cuya implementación ha sido necesaria para el sistema desarrollado y que conforman el software de este.

El séptimo capítulo, Implantación, abarca el proceso de despliegue del sistema, necesidades técnicas y dependencias software. Detalla el proceso de paso a producción y planes de formación.

El octavo capítulo, Evaluación, establece el plan de pruebas realizado para asegurar la calidad y correcto funcionamiento del sistema desarrollado, a la vez que presenta un estudio de satisfacción de los usuarios frente al producto obtenido.

El noveno y último capítulo, Conclusiones y trabajos futuros, incluye una serie de conclusiones técnicas y personales del sistema desarrollado, así como trabajos futuros pensados para ampliar la funcionalidad del mismo.

Por último, al final del documento se incluyen como apéndices los acrónimos y el glosario utilizados.

Estado de la cuestión

Situación inicial e infraestructura heredada

Partimos de una infraestructura base, la cual está formada por **cuatro aulas docentes** (dos en el edificio Torres Quevedo y dos en el edificio Sabatini) y un **rack con servidores propios del LDI en el CPD**. En este rack de servidores se encuentra un servidor dedicado, ***instalador***, que se utiliza como **punto centralizado de gestión**. Desde este servidor se comprueba el estado de la infraestructura, se lanzan tareas en remoto, y en definitiva se utiliza para las labores de control y mantenimiento diario de la infraestructura del LDI.

Otros servidores destacables del rack del LDI son ***web*** y ***ultraheroe***. Web sirve la **página web del LDI** alojada en `https://www.lab.inf.uc3m.es` con ayuda de un servidor Apache [1]. Ultraheroe es utilizado principalmente como un **servidor de respaldo**. Contiene copias de seguridad y las herramientas necesarias para replicar la infraestructura en caso de catástrofe. Asimismo, es utilizado como **entorno de producción** para aplicaciones **no críticas**.

Como infraestructura para el desarrollo e implementación del sistema el LDI cede tres minicomputadores **Raspberry Pi modelo 3B+** [2], un único módulo de sensores ***SenseHat*** [3], un módulo de **cámara**, dos **monitores** y una **bombilla Wi-Fi** Philips® Hue White and Color Ambiance. Estos minicomputadores cuentan con las siguientes características a destacar:

- Procesador ARM de cuatro núcleos y 64 bits a 1.4GHz.
- 1GB de memoria LPDDR2 SDRAM.
- Disco de sistema en tarjetas micro SD.
- Conectividad de red IEEE 802.11.b/g/n/ac y Gigabit Ethernet.
- Salida de vídeo/audio HDMI.
- Potencia de entrada de 5V a 2.5A.

Tras haber definido anteriormente los Objetivos y alcance del proyecto se procede al estudio de alternativas.

Estudio de alternativas y selección de base tecnológica

Para el estudio de alternativas y selección de base tecnológica se diferencian **dos subsistemas**: el de monitorización del estado del **CPD** y servidores, y el de monitorización de las **aulas** y equipos.

Subsistema de monitorización del estado del CPD y servidores

Primero se atiende al subsistema del CPD. Los técnicos del LDI intentaron monitorizar los servidores hace algunos años. Para ello, implementaron una **instancia de Zabbix** [4] que se me solicita que **recupere**, ya que quieren recuperar el esfuerzo invertido y centralizar desde ahí algunas alertas por correo electrónico. Sin embargo, coinciden conmigo en que la interfaz de Zabbix no era la más adecuada para mostrar métricas relativas a los servidores por ser **poco intuitiva**.

Así, se valoran varias alternativas para **proyección de datos** como son Grafana [5], Kibana [6] y Keen dashboard [7]. Una implementación propia de un panel web queda descartada pues se requiere proyectar gráficos sencillos y ya **existen muchas herramientas** destinadas a ello.

Cualquiera que conozca las herramientas mencionadas se habrá percatado que son alternativas de **software libre** o código abierto. Esto es así para **facilitar** el entendimiento, mantenimiento e implementación de las herramientas con la infraestructura existente y sin coste económico.

	Grafana	Kibana	Keen dashboard
Licencia	Apache	Apache & custom (open)	MIT
Coste de puesta en marcha	Muy simple	Muy costoso	Medio
Documentación	Abundante y bien redactada, paso a paso	Abundante pero compleja	Buena
Comunidad y soporte	Comunidad activa y facilidad para conseguir soporte	Comunidad activa y facilidad para conseguir soporte	Comunidad pequeña
Integración con Zabbix	Sí, mediante plugins de terceros [8]	No, requiere que los datos estén en la suite elasticsearch	"Sí", pero requiere extraer los datos de Zabbix primero
Funcionalidad y extras	Básica	Elevada	Básica
Flexibilidad	Limitada, pero existen plugins de terceros	Limitada para herramientas fuera de la suite de elasticsearch	Completa
Complejidad y mantenimiento	Simple	Muy complejo	Depende de cómo se extraigan los datos de Zabbix

Tabla 1. Comparativa alternativas dashboards.

En la tabla 1 podemos apreciar una comparativa entre las tres herramientas en base a varias características. Se han resaltado en **verde** los valores que se consideran muy buenos o deseables, en **amarillo** aquellos que podrían suponer dificultades o limitaciones durante el desarrollo o el mantenimiento, y en **rojo** valores no deseables o problemáticos.

A simple vista debe sobresalir que Kibana es rechazada por varios motivos, uno de los cuales es su **coste de implementación** y mantenimiento. Sin embargo, el aspecto definitivo para su rechazo es **no poder aprovechar** los datos recogidos por **Zabbix**.

Grafana se impone a Keen debido a la facilidad para obtener los datos con solo instalar un plugin de terceros [8], lo que facilitará enormemente su mantenimiento. Asimismo, una mayor **comunidad y una documentación** abundante y detallada siempre es un factor clave a la hora de elegir una herramienta de software libre.

Para la **obtención de datos** sobre el estado del CPD (temperatura de la sala, humedad, etc.) se utilizará una de las **Raspberry Pi** donadas por el LDI aprovechando los **módulos** de cámara y el de sensores (*SenseHat*). Dado que solo se requiere obtener valores simples se considera la programación de unos **scripts** que soliciten la información al hardware. Se estudian las siguientes alternativas de lenguajes de programación para ser implementados:

	Bash	Python	Go
Conocimiento del lenguaje	Medio	Muy alto	Alto
Documentación del lenguaje	Abundante y bien redactada	Abundante y bien redactada	Abundante y bien redactada
Comunidad del lenguaje y soporte	Comunidad activa y facilidad para conseguir soporte	Comunidad activa y facilidad para conseguir soporte	Comunidad activa y facilidad para conseguir soporte
Compatibilidad[1]	Accesible desde los dispositivos en `/dev`	Existencia de librería oficial de Raspberry Pi para controlar los módulos	Existencia de librerías de terceros para controlar los módulos
Orientación a scripts	Sí	Sí	No, pero permite ejecución interpretada
Complejidad de implementación	Compleja, requiere entender cómo es el dispositivo para el SO	Solo se requiere importar las funciones de la librería	Solo se requiere importar las funciones de la librería
Rendimiento	Muy bueno	Bueno	Muy bueno interpretado. Óptimo si se compila

[1] Facilidad para trabajar con los módulos de hardware de la Raspberry Pi desde cada lenguaje de programación.

Tabla 2. Comparativa alternativas lenguajes de programación para desarrollar los scripts.

Tras la comparativa de la tabla 2 se concluye que **el lenguaje más adecuado** para codificar los scripts **es python**, ya que cumple con buena nota en todos los apartados y especialmente por ser el único lenguaje con **soporte oficial para los módulos de Raspberry Pi**. La versión del lenguaje a utilizar será la **versión 3** en su última actualización estable, ya que la versión 2 de Python ha sido desaprobada y perderá todo mantenimiento el próximo año 2020 [9].

Subsistema de monitorización de las aulas y equipos

El subsistema de aulas requiere **comunicar varios componentes y centralizar la información**. Por un lado, se requiere obtener la **información de las reservas de aulas** docentes, las cuales se encuentran en una base de datos del LDI situada en un servidor específico. La misma información en formato *human readable* es **accesible desde** un apartado de **la página web del LDI** [10].

Por otro lado, la **información** acerca del **estado de los equipos** es **accesible** a través del servidor ***instalador*** que ya se ha mencionado al inicio de este capítulo. Esta información se obtiene mediante la ejecución de unos **scripts en python realizados por Lázaro** W. Fernández, un becario anterior del LDI y sobre los cuales se puede obtener más información consultando su TFG [11].

Estos scripts ya estaban **implementados antes** de plantear el desarrollo de este sistema. Asimismo, cuando se desarrollaron se pensaron para ser ejecutados manualmente y para que su **salida** fuera **interpretada por humanos**, no por otros sistemas.

Se considera que la forma más sencilla de comunicar, centralizar y coordinar todos estos procesos y sistemas es mediante una **API web**. Esto es debido a que los servidores van a necesitar comunicarse y coordinarse por la red y una API web permite esto de **forma sencilla** requiriendo solo el uso del protocolo HTTP.

Entre los lenguajes para desarrollar la/s API REST propuesta/s se estudian:

	JavaScript (nodeJS)	Python	Go
Conocimiento del lenguaje	Medio	Muy alto	Alto
Documentación del lenguaje	Abundante y bien redactada	Abundante y bien redactada	Abundante y bien redactada
Comunidad del lenguaje y soporte	Comunidad activa y facilidad para conseguir soporte	Comunidad activa y facilidad para conseguir soporte	Comunidad activa y facilidad para conseguir soporte
Utilidades de red	Orientado a web. Existencia de *frameworks* para API REST	No orientado a web, pero existen frameworks con peso para API REST	Orientado a red y microservicios web. Soporte nativo para API REST
Complejidad de implementación	Depende de la arquitectura del framework usado	Depende de la arquitectura del framework usado	Uso de la librería estándar. Soporte nativo
Rendimiento	Mejorable	Bueno	Muy bueno interpretado. Óptimo si se compila
Sobrecarga del sistema (dependencias)	Necesidad de instalar gran cantidad de dependencias (frameworks)	Gran parte del desarrollo se puede realizar sin muchas dependencias	Soporte nativo en la biblioteca estándar, se pueden evitar las dependencias.

Tabla 3. Comparativa alternativas lenguajes de programación para desarrollar la/s API REST.

Como se puede observar en la tabla 3, JavaScript es descartado por ser el lenguaje en el que se tiene menos experiencia y en especial por la **sobrecarga que supondría al sistema en rendimiento** y dependencias; un sistema que se pretende implementar en un **hardware de bajas prestaciones** como son las Raspberry Pi.

Entre Python y Go, **la elección de Go** es obvia por el mayor soporte y rendimiento que brindará a nuestro sistema. El que sea un **lenguaje orientado a servicios web** y permita desarrollar nuestro sistema con facilidad solo con las **utilidades de la librería estándar** es un punto decisivo a favor, ya que lo hace idóneo para esta tarea. Asimismo, la compilación permitirá un **mayor rendimiento** frente a python y la posibilidad de **evitar la instalación de dependencias** si se usasen, pues irán embebidas en el binario.

Se plantea **un panel web** o ***dashboard*** que permita de un vistazo asimilar toda la información relativa al estado de las aulas y los equipos. El subsistema requiere una **interfaz muy especializada** que es imposible de obtener con soluciones ya desarrolladas como las estudiadas en la tabla 1, por lo que el **desarrollo de una alternativa propia es inevitable**.

Al tratarse de un desarrollo de una interfaz de usuario web, el uso de tecnologías como HTML, CSS y en especial JavaScript es prácticamente inevitable y es de hecho la mejor opción. Así pues, el estudio de alternativas se centra en los distintos **frameworks de JavaScript** (nodeJS) que se pueden utilizar para el desarrollo de entre los siguientes: Angular[2] [12], React [13] y Vue.js [14].

	Angular	React	Vue.js
Licencia	MIT	MIT	MIT
Desarrollador / principal impulsor	Google	Facebook	Ex-empleado de Google
Conocimiento del framework	Básico	Ninguno	Ninguno
Documentación del framework	Abundante y bien redactada	Abundante y bien redactada	Algunos aspectos no están del todo documentados
Curva de aprendizaje	Promedio	Sencilla	La más sencilla
Comunidad del framework y soporte	Comunidad activa y facilidad para conseguir soporte	Comunidad **muy** activa y facilidad para conseguir soporte	Comunidad activa pero más pequeña que las otras alternativas
Adopción en la industria[3]	21.0 %	78.1 %	0.8 %
Versatilidad	Versátil	Muy versátil	Un poco inmaduro, lo que le hace perder versatilidad
Escalabilidad	Escalable	Fácilmente escalable	Un poco inmaduro, lo que le hace ser difícilmente escalable

[3] Valores obtenidos del porcentaje de ofertas de empleo para cada framework extraídos de Indeed.com [15].

Tabla 4. Comparativa alternativas frameworks JavaScript para desarrollar el dashboard.

Estudiadas las alternativas en la tabla 4, si bien se tiene poco conocimiento de los frameworks y es vue.js el que tiene la curva de aprendizaje más sencilla, este es descartado dadas sus **grandes carencias** frente a las otras dos alternativas.

Así pues, entre Angular y React, **se impone React** por su mayor comunidad, soporte, y apoyo de la industria; y dado que posee mayor versatilidad y escalabilidad que Angular.

[2] Si bien Angular no es un framework escrito en JavaScript, sino en TypeScript, las diferencias son mínimas.

Para facilitar el desarrollo así como simplificar el despliegue en las Raspberry Pi y asegurar un mayor rendimiento se valora la posibilidad de utilizar un **generador de sitios/aplicaciones web** como una herramienta más en la implementación de la solución. De las alternativas basadas en React se estudian las tres más populares: GatsbyJS [16], Next.js [17] y Create React App [18].

	GatsbyJS	Next.js	Create React App
Licencia	MIT	MIT	MIT
Conocimiento de la herramienta	Nulo	Nulo	Nulo
Documentación de la herramienta	Abundante y bien redactada	Suficiente. Bien redactada	Abundante y bien redactada
Curva de aprendizaje	Sencilla, similar a la propia curva de React	Algo complicada al inicio	Sencilla, similar a la propia curva de React
Comunidad de la herramienta y soporte	Comunidad grande y activa. Facilidad para conseguir soporte	Comunidad grande pero poco activa. Dificultad para conseguir soporte	Comunidad **muy** grande y activa. Facilidad para conseguir soporte
Rendimiento	Muy bueno, compilado a estáticos	Recae toda la carga sobre la raspberry (server side rendering)	Bueno. Recae sobre el usuario (client side rendering)

Tabla 5. Comparativa alternativas generadores de contenido estático para React.

De la comparativa mostrada en la tabla 5 se deduce el rechazo a utilizar Next.js, pues su peor **rendimiento** así como una **curva complicada de aprendizaje** unida a la dificultad para obtener soporte la descarta como alternativa viable.

Así pues, la elección es entre GatsbyJS y Create React App. La herramienta elegida es **GatsbyJS**, que se impone por su mejor **rendimiento**.

Finalmente, siguiendo en la línea de simplificar el despliegue en las Raspberry Pi y asegurar un mayor rendimiento, se decide desarrollar un **servidor web minimalista** para servir el contenido. Se implementará en el **lenguaje Go** por los mismos motivos que los expuestos tras el estudio realizado en la comparativa de la tabla 3 para las API REST, y dado que este desarrollo se prevee que no conlleve más de 10 líneas de código.

Marco regulador

Análisis de la legislación aplicable

Durante el desarrollo del proyecto se ha debido hacer frente a ciertos **aspectos legales** derivados de la solución adoptada para obtener parte de la funcionalidad del sistema.

Dado que no contamos con un sensor fotosensible, se pretende utilizar la **cámara** de la Raspberry Pi para **monitorizar** si **la luz** dentro del CPD está encendida o no. Esto requiere tomar **imágenes** de la sala en las que con gran seguridad aparecerían las **personas** que allí se encontrasen.

Estas fotografías en las que aparecieran personas supondrían datos personales dado que se trata de «información sobre una persona física identificada o identificable», según el **artículo 4.1 del eglamento General de Protección de Datos (RGPD)** [19].

Si bien no se almacenarían las imágenes sino que se realizaría un tratamiento en tiempo real de la luminosidad de los píxeles de la misma, es de igual aplicación el RGPD ya que se estaría realizando un tratamiento de datos al aplicar técnicas informáticas sobre la imagen. Así, la citada norma define en su **artículo 4.2** tratamiento como «**cualquier operación** o conjunto de operaciones realizadas sobre datos personales o conjuntos de datos personales, ya sea por procedimientos automatizados o no, como la recogida, registro, organización, estructuración, conservación, adaptación o modificación, extracción, consulta, utilización, comunicación por transmisión, difusión o cualquier otra forma de habilitación de acceso, cotejo o interconexión, limitación, supresión o destrucción».

Para evitar entrar en **materia legal innecesaria para el fin que buscamos** se propone adoptar una solución que evite la captura y por lo tanto el tratamiento de estos datos personales, puesto que no necesitamos que en las imágenes aparezcan personas. Para ello, la cámara se dispondrá **apuntando hacia la pared y a unos 1-2 centímetros** de la misma. Así, se capturará únicamente la imagen de una pared blanca cuyo brillo se utilizará para calcular si la luz del CPD se encuentra encendida o apagada.

Estándares técnicos

Este proyecto se ve afectado principalmente por los siguientes estándares técnicos:

- **ISO/IEC 29110** que regula los procesos de ciclo de vida del software.
- **ISO/IEC 25000** que regula la evaluación de la calidad del software.
- **IEEE 802** que regula la red de área local (LAN) utilizada para la comunicación de los equipos.

Cuestiones de propiedad intelectual

Existen diversas **cuestiones de propiedad intelectual** que afectan al desarrollo del sistema así como al producto obtenido.

En cuanto al desarrollo del sistema, se ha utilizado software, tecnologías y lenguajes de programación con licencias de **software libre** que cumplen las cuatro libertades establecidas por GNU y la Free Software Foundation (FSF) [20].

Sobre las **tecnologías y software de terceros**, ya se ha mencionado en el apartado Estudio de alternativas y selección de base tecnológica que todas las herramientas poseen una **licencia de software libre permisiva**. Todas poseen licencia MIT [21] salvo Grafana que está licenciado bajo Apache [22], y ambas licencias permiten el uso comercial, uso privado, distribución y modificación del software.

Los **lenguajes** utilizados (Python, Go, JavaScript) se encuentran asimismo licenciados bajo **licencias permisivas de software libre**. Así, Python se encuentra licenciado bajo una licencia propia (Python Software Foundation License o PSFL) libre y compatible con GPL [23]. Go a su vez está licenciado bajo la licencia BSD [24] de software libre, una licencia muy permisiva al estilo de la licencia MIT. JavaScript es una marca registra de Oracle Corporation [25], pero se permite su uso sin restricción para el desarrollo de aplicaciones y sitios web.

Respecto al producto desarrollado, la legislación nacional establece en el **Real Decreto Legislativo 1/1996 del 12 de abril** [26], **artículo 10.1**, que «Son objeto de propiedad intelectual todas las creaciones originales [...] comprendiéndose entre ellas [...] Los programas de ordenador». Asimismo, este RD define en su **artículo 96.1** "programas de ordenador" como «toda secuencia de instrucciones o indicaciones destinadas a ser utilizadas, directa o indirectamente, en un sistema informático para realizar una función o una tarea o para obtener un resultado determinado, cualquiera que fuere su forma de expresión y fijación». Así pues, es de aplicación la **Ley de Propiedad Intelectual** de ámbito nacional.

De igual manera es de aplicación la normativa de la Unión Europea en materia de derechos de autor al ser España un país miembro de la Unión. **La Directiva 2009/24/CE del Parlamento Europeo y del Consejo** [27], que regula la protección jurídica de programas de ordenador, establece que:

- «los Estados miembros protegerán mediante derechos de autor los programas de ordenador como obras literarias tal como se definen en el Convenio de Berna para la protección de las obras literarias y artísticas.» (**artículo 1.1**).
- «Se considerará autor del programa de ordenador a la persona física o grupo de personas físicas que lo hayan creado o, cuando la legislación de los Estados miembros lo permita, a la persona jurídica que sea considerada titular del derecho por dicha legislación.» (**artículo 2.1**).
- «Cuando un trabajador asalariado cree un programa de ordenador en el ejercicio de las funciones que le han sido confiadas, o siguiendo las instrucciones de su empresario, la titularidad de los derechos económicos correspondientes al programa de ordenador así creado corresponderán, exclusivamente, al empresario, salvo pacto en contrario.» (**artículo 3**).

- «los derechos exclusivos del titular [...] incluirán el derecho de realizar o de autorizar: a) la reproducción total o parcial de un programa de ordenador por cualquier medio y bajo cualquier forma, [...] b) la traducción, adaptación, arreglo y cualquier otra transformación de un programa de ordenador [...] c) cualquier forma de distribución pública» (**artículo 4.1**).

Por ello, como autor de los derechos del software y tras tratar este asunto con el personal del LDI, he tomado la **decisión de publicar el software** y demás documentación asociada al mismo que no comprometa el correcto y seguro funcionamiento del servicio del LDI en un repositorio de GitHub [28] bajo una **licencia MIT** [21] **de software libre**. El repositorio actualmente se encuentra privado y será liberado públicamente tras la publicación de esta memoria en la biblioteca de la universidad.

Entorno socio-económico

Metodología utilizada y ciclo de vida

Durante el desarrollo del sistema se ha utilizado una **metodología de desarrollo** de software **ágil** o *agile*. En concreto se ha seguido la metodología **Kanban** [29].

Kanban es una metodología **muy fácil de utilizar, actualizar y entender**; ya que se apoya principalmente en el uso de un tablero Kanban de tarjetas visuales, lo que proporciona una **gestión de tareas muy visual**. Se basa en cuatro principios fundamentales:

- **Calidad garantizada.** Todo lo que se hace debe salir bien a la primera. En la metodología Kanban no se premia la rapidez (como podría ser en scrum y otras metodologías ágiles), sino la calidad final de las tareas. Este principio se basa en el hecho de que muchas veces cuesta más arreglar las tareas que se han hecho deprisa que hacerlo bien a la primera.
- **Principio YAGNI (You Aren't Gonna Need It).** No se debe agregar funcionalidades que no van a ser necesarias, pues es malgastar recursos y abrir la puerta a errores.
- **Flexibilidad y mejora continua.** Las tareas se pueden modificar o priorizar en cualquier momento, dando flexibilidad a la planificación.
- **Stop starting, start finishing.** Se debe priorizar el trabajo en curso y la finalización de tareas abiertas en lugar de empezar nuevas tareas.

Se ha considerado que esta metodología es la adecuada para este proyecto por varios motivos:

- La metodología Kanban permite una **entrega y mejora continua**, permitiendo que el cliente vea en todo momento el desarrollo del producto y se involucre en el mismo con comentarios y propuestas de mejora, asegurando así que el producto final obedece a sus intereses.
- La utilización de un tablero de tarjetas Kanban permite en todo momento **conocer de un vistazo el ritmo y fases de trabajo actuales** así como el trabajo que aún queda por hacer, lo que ayuda a seguir la planificación del proyecto.
- Es una metodología **sencilla de utilizar** que puede ser utilizada **por una única persona**, como en este caso.

Se ha utilizado un tablero digital creado en la aplicación Trello [30]. El tablero se ha definido como se muestra en la figura 1, con seis columnas. Estas son:

- **Ideas locas.** Sugerencias de requisitos propuestos por el cliente como extras de la funcionalidad original requerida y con muy baja prioridad.
- **Por hacer.** Requisitos confirmados o tareas pendientes de cualquier fase del ciclo de vida.
- **En proceso.** Tareas actualmente en ejecución. Se ha establecido que no puede haber más de tres tareas abiertas al mismo tiempo en esta columna.
- **Revisar.** Tareas completadas de las que hay que revisar algún aspecto. Se ha establecido que no puede haber más de tres tareas abiertas al mismo tiempo en esta columna.
- **Hecho.** Tareas completadas.
- **Descartado.** Tareas descartadas por cualquier motivo. Normalmente las tarjetas que aquí se encuentran provienen directamente de las dos primeras columnas.

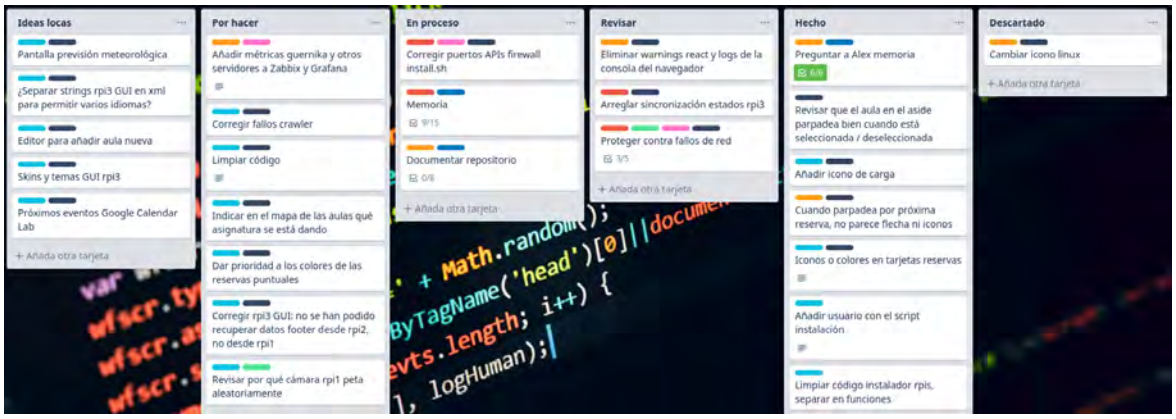

Figura 1: Tablero visual Trello utilizado para gestionar el ciclo de vida del proyecto. Captura de pantalla.

Las tarjetas tienen asociadas etiquetas de distintos colores que indicandan a qué fase del ciclo de vida pertenecen, con qué subsistema se corresponden, su prioridad y tipo.

Los lunes de cada semana durante el desarrollo se presenta a los técnicos del LDI (el cliente) los avances realizados, el producto en su versión actual, y el estado actual del flujo de trabajo y tareas. En **reuniones ligeras de 10-20 minutos** se negocia la prioridad de las tareas pendientes y se preparaba el flujo de trabajo de la semana, de acorde por supuesto a la Planificación del proyecto definida más adelante.

Dado que el proyecto ha sido desarrollado siguiendo una metodología *agile*, **en contraposición con un desarrollo tradicional en cascada**, los requisitos del sistema a desarrollar han ido cambiando continuamente con el tiempo. Asimismo, el proyecto estaba en **diversas fases a la vez**, llegando a estar algunos componentes en fase de análisis mientras que otros se encontraban en diseño, implantación o evaluación. Por ello, en los siguientes capítulos se presentan los requisitos, diseños, pruebas, etc. **finales** obtenidos tras el desarrollo del producto (a excepción de los requisitos de usuario, que esos son los planteados originalmente).

Planificación

La planificación del tiempo dedicado al proyecto se ha realizado teniendo en cuenta la asignación de créditos ECTS que corresponde al TFG y la fecha de entrega de la memoria del mismo. El TFG supone 12 créditos ECTS, lo cual estima **360 horas de trabajo** (30 horas/ECTS x 12 ECTS). La **fecha límite** de entrega de la memoria finaliza el **17 de junio** de 2019.

Dado que este proyecto tiene su alcance dentro del torno del LDI, requiere de la infraestructura del mismo, y va a ser de aprovechamiento para el personal del LDI tras su desarrollo; el trabajo en el proyecto se realizará durante las horas de trabajo de mis prácticas extracurriculares en el LDI. Dado que el contrato estipula una jornada de **20 horas semanales** se preveen necesarias 360 / 20 = **18 semanas** para el desarrollo del proyecto.

Sin embargo, debido al resto de mis obligaciones durante el normal desarrollo de las prácticas extracurriculares, es de prever que no se dispondrá de las 20 horas cada semana para el desarrollo del proyecto. Así pues, se añade una semana extra a la planificación resultando esta en un total de **19 semanas**. La planificación final, que se ha podido seguir sin problemas ni modificaciones, queda reflejada en el diagrama Gantt de la figura 2.

Dado que para el desarrollo del sistema se utiliza una **metodología *agile*** se pueden apreciar algunas peculiaridades en el diagrama Gantt con respecto a un desarrollo tradicional en cascada como pudiera ser Métrica v3. Se puede observar que las fases de análisis y diseño se solapan una semana. Esto es porque la metodología actual permite empezar a trabajar en el diseño del sistema mientras aún se están refinando los requisitos del mismo. Se consideró que comenzar a definir la arquitectura del sistema cuanto antes ayudaría a repasar y reformular los requisitos del sistema.

La fase de análisis tiene un mayor peso que diseño, ya que gran parte del trabajo de diseño se realizará durante la fase de implementación y desarrollo del código del sistema, pues es un **proceso de entrega continua con constante interacción con el cliente**. Asimismo, la importancia de las fases de análisis y diseño es reducida respecto a lo habitual, superando la fase de implementación la duración de las dos anteriores en conjunto. Esto nuevamente se debe a la naturaleza del proceso de trabajo, pues la implementación está sujeta a continuo cambio y reformulación debido a las interacciones con el cliente, realizándose en la misma también labores de análisis y diseño.

Por último, debido a la entrega continua durante la implementación, **la fase de implantación es difusa** y se fusiona con esta. Para **asegurar la consistencia del sistema desplegado** se establece una semana para realizar una prueba de implantación y despliegue desde cero. Asimismo la ejecución del plan de pruebas durante la fase de evaluación solo sirve para detectar y corregir problemas menores que **aseguren la calidad del sistema**, pues la mayoría de errores se han corregido durante la fase debido a su naturaleza de entrega continua.

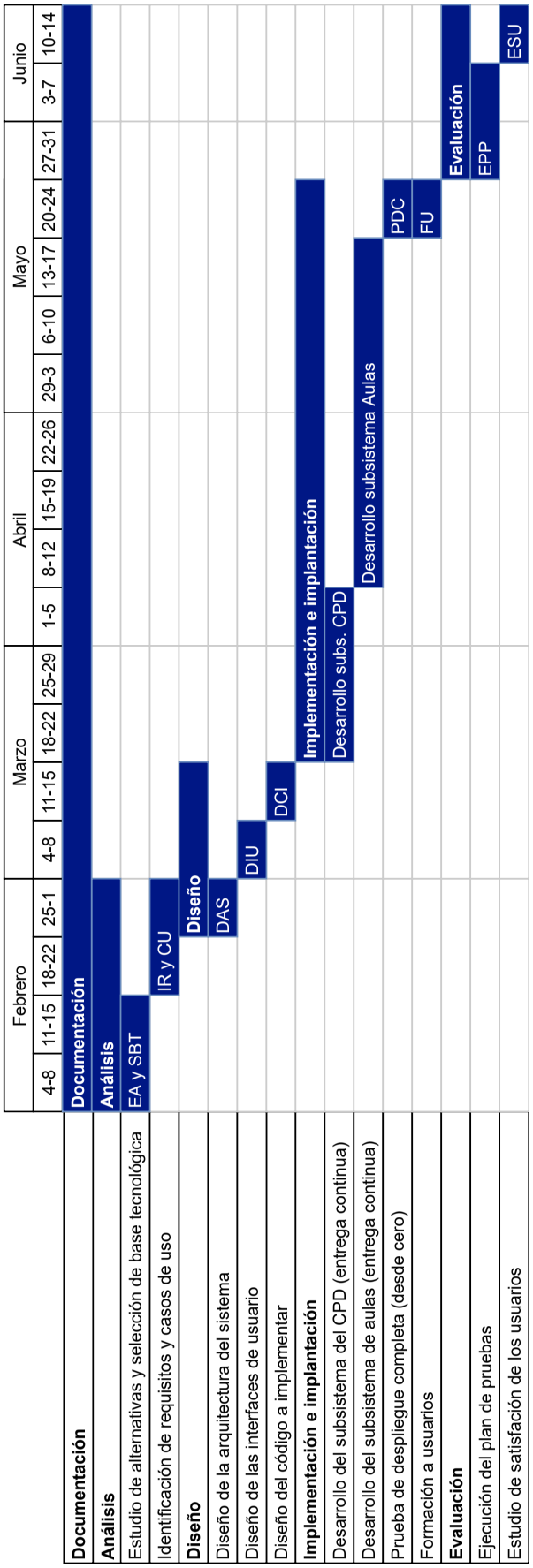

Figura 2: Diagrama Gantt de la planificación del proyecto. Elaboración propia.

Presupuesto

El presupuesto del sistema se ha calculado en base a los recursos planificados y el coste de personal de los distintos roles identificados en el proceso. Para el coste por hora del personal se ha comprobado el salario medio de cada rol [31] por año y traducido a salario por hora [32] para una jornada de 40 horas semanales y 52 semanas trabajadas al año. La asignación de horas de cada rol se realiza conforme a la planificación del apartado anterior (20 horas por semana).

En la tabla 6 se presenta el **coste del personal** con cargo al proyecto.

COSTE DE PERSONAL CON CARGO AL PROYECTO			
Rol	**Salario por hora (€)**	**Horas**	**Total (€)**
Analista	11.99	80	959.20
Diseñador	15.28	60	916.80
DevOps	20.86	200	4 172.00
Asegurador de la calidad	15.00	60	900.00
		TOTAL	**6 948.00 €**

Tabla 6. Presupuesto: coste de personal con cargo al proyecto.

En la tabla 7 se presenta el **coste de software**. Dado que se utilizan tecnologías de software libre, el coste software es cero.

COSTE DE SOFTWARE CON CARGO AL PROYECTO			
Concepto	**Coste por unidad (€)**	**Unidades**	**Total (€)**
		TOTAL	**0.00 €**

Tabla 7. Presupuesto: coste de software con cargo al proyecto.

En la tabla 8 se presenta el **coste de hardware** del proyecto.

COSTE DE HARDWARE CON CARGO AL PROYECTO			
Concepto	**Coste por unidad (€)**	**Unidades**	**Total (€)**
Minicomputador Raspberry Pi modelo 3B+	39.99	3	119.97
Fuente de alimentación Raspberry Pi	12.99	3	38.97
Tarjetas de memoria micro SD 32 GB	8.99	3	26.97
Módulo de Raspberry Pi – cámara	23.99	1	23.99
Módulo de Raspberry Pi – SenseHat	39.99	1	39.99
Monitor 23 pulgadas con HDMI y altavoces	150.00	2	300.00
Cable HDMI	6.29	2	12.58
Cable Ethernet 10 metros	10.50	4	42.00
Kit bombilla Wi-Fi Philips® Hue + Hue Bridge	109.85	1	109.85
		TOTAL	**714.32 €**

Tabla 8. Presupuesto: coste de hardware con cargo al proyecto.

El **coste final del proyecto** queda definido en la tabla 9.

COSTE FINAL DEL PROYECTO	
Concepto	**Coste (€)**
Costes de personal	6948.00
Costes de software	0.00
Costes de hardware	714.32
SUBTOTAL	**7 662.32 €**
Margen de riesgo (5 %)	383.12
Margen de beneficio (10 %)	766.23
TOTAL	**8 811.67 €**

Tabla 9. Presupuesto: coste final del proyecto.

Con lo que el coste final del proyecto será de **ocho mil ochocientos once euros con sesenta y siete céntimos**. Este presupuesto no incluye impuestos, pues son desgravables. Tampoco incluye mantenimiento ya que al usarse tecnologías libres y el cliente poseer perfil técnico se asume que este proporcionará su propio servicio de mantenimiento.

Impacto socio-económico

Con la implantación del sistema se espera que los técnicos del laboratorio dispongan de más tiempo para realizar el resto de sus obligaciones profesionales, pues parte del mismo estará monitorizando activamente aspectos que antes se monitorizaban de manera manual y alertando de cualquier problema. Asimismo el sistema facilitará y reducirá los tiempos de acceso a información acerca del estado de la infraestructura. Se prevé así un **mayor aprovechamiento de los recursos** humanos del LDI.

Monitorizar y conocer en todo momento las condiciones ambientales del CPD, concretamente de humedad y temperatura, permite regular adecuadamente las máquinas refrigeradoras de la sala reduciendo al mínimo su negativo **impacto medioambiental** y la huella de emisiones de carbono de la climatización de la sala. Diversos investigadores y grupos ecologistas ya han avisado, tras varios estudios, que el uso no controlado de los aparatos de aire acondicionado puede provocar una subida de la temperatura habitual en verano de hasta 2ºC [33]. Además, con la regulación adecuada de la refrigeración este servicio contribuirá de una forma más activa al cumplimiento de los objetivos de sostenibilidad ambiental definidos en el apartado 5.6.8 del *Plan Estratégico 2016-2022* [34] elaborado por nuestra Universidad.

Asimismo, una correcta regulación de las máquinas refrigeradoras reducirá el consumo eléctrico de las mismas al mínimo necesario, **minimizando** a su vez el **coste económico** derivado de la **factura eléctrica**. Este supone uno de los principales si no el principal coste de los centros de procesamiento de datos de cualquier entidad o institución. Del mismo modo, es conveniente desta-

car que el uso de minicomputadores Raspberry Pi reduce enormemente los costes de compra de la infraestructura y de su mantenimiento frente a los servidores físicos tradicionales, así como su consumo y factura eléctrica.

Igualmente, monitorizar si la luz del CPD está encendida evita que se quede encendida por descuidos u olvidos, reduciendo al igual que para la climatización la **factura eléctrica** derivada de este consumo, si bien en este caso es menos notorio que para las máquinas de aire acondicionado.

A su vez, la utilización de tecnologías de software libre permite **evitar costes** en licencias así como reducir o eliminar los costes de servicio técnico de terceros.

De la misma forma, la utilización de estas tecnologías libres así como la publicación de todo el código y software desarrollado [28] bajo una licencia MIT [21] de software libre responden a un **objetivo ético**, contribuyendo a fomentar la transparencia del software, la colaboración en su desarrollo y seguridad, la compartición de conocimiento y la libertad de uso, modificación y redistribución del mismo; devolviendo así a la comunidad un esfuerzo aprovechable por y para la misma.

Análisis

Requisitos de usuario

Los requisitos **originales** de usuario se presentan en forma tabular siguiendo la siguiente plantilla:

Identificador			
Título			
Prioridad		**Necesidad**	
Verificabilidad		**Estabilidad**	
Descripción			

Tabla 10. Plantilla requisitos de usuario.

Cada uno de los campos de la tabla 10 puede tomar los siguientes valores:

- **Identificador:** código que identifica unívocamente al requisito de usuario. Será de la forma *RU-XX*, donde *RU* son las siglas para *Requisito de Usuario* y *XX* representa el identificador numérico del requisito (01-99).
- **Título:** descripción breve del requisito.
- **Prioridad:** indica cómo de prioritario es el actual requisito para el usuario respecto a los demás, y por tanto la importancia que se debe dar a su pronta finalización. Puede tomar los valores *alta*, *media* y *baja*.
- **Necesidad:** indica cómo de deseable es el actual requisito para el usuario. Puede tomar los valores *esencial*, *deseable* y *opcional*.
- **Verificabilidad:** indica en qué grado es posible comprobar si el actual requisito se ha implementado correctamente para el usuario. Puede tomar los valores *alta*, *media* y *baja*.
- **Estabilidad:** indica si es **probable** que el actual requisito sea modificado a lo largo del desarrollo del sistema. Dado que se ha seguido una metodología agile, es **posible** la modificación de requisitos durante cualquier fase del desarrollo del sistema. Los requisitos **de usuario** aquí recogidos son los originales planteados por los usuarios inicialmente. Puede tomar los valores *estable* o *inestable*.
- **Descripción:** descripción extensa y detallada de las demandas del usuario para satisfacer el actual requisito.

Los requisitos planteados inicialmente por el cliente y a la vez futuros usuarios (técnicos del LDI) son los siguientes:

RU-01			
Título	Visualización de la temperatura dentro del CPD.		
Prioridad	Alta	**Necesidad**	Esencial
Verificabilidad	Alta	**Estabilidad**	Estable
Descripción	El sistema debe monitorizar la temperatura actual en el interior del CPD y su valor debe poder reconocerse al momento si se observa la interfaz del sistema.		

Tabla 11. RU-01: Visualización de la temperatura dentro del CPD.

RU-02			
Título	Visualización de la humedad dentro del CPD.		
Prioridad	Alta	**Necesidad**	Esencial
Verificabilidad	Alta	**Estabilidad**	Estable
Descripción	El sistema debe monitorizar la humedad actual en el interior del CPD y su valor debe poder reconocerse al momento si se observa la interfaz del sistema.		

Tabla 12. RU-02: Visualización de la humedad dentro del CPD.

RU-03			
Título	Visualización del nivel de carga del SAI.		
Prioridad	Alta	**Necesidad**	Esencial
Verificabilidad	Alta	**Estabilidad**	Estable
Descripción	El nivel de carga del SAI del rack de servidores del LDI debe poder reconocerse al momento si se observa la interfaz del sistema.		

Tabla 13. RU-03: Visualización del nivel de carga del SAI.

RU-04			
Título	Visualización del estado de la iluminación del CPD.		
Prioridad	Media	**Necesidad**	Deseable
Verificabilidad	Alta	**Estabilidad**	Estable
Descripción	El sistema debe monitorizar e indicar de algún modo si la luz del interior de la sala del CPD se encuentra encendida o no.		

Tabla 14. RU-04: Visualización del estado de la iluminación del CPD.

RU-05			
Título	Alerta de problemas y situaciones anómalas en el CPD.		
Prioridad	Alta	**Necesidad**	Esencial
Verificabilidad	Alta	**Estabilidad**	Estable
Descripción	El sistema debe informar mediante alertas reconocibles ante situaciones extrañas, anómalas o problemáticas en el CPD o el rack de servidores del LDI.		

Tabla 15. RU-05: Alerta de problemas y situaciones anómalas en el CPD.

RU-06			
Título	Integración de las métricas del CPD con Zabbix.		
Prioridad	Media	**Necesidad**	Esencial
Verificabilidad	Alta	**Estabilidad**	Inestable
Descripción	Todas las métricas del CPD o el rack de servidores del LDI recogidas por el sistema deben ser enviadas a Zabbix como requisito de integración con la infraestructura heredada.		

Tabla 16. RU-06: Integración de las métricas del CPD con Zabbix.

RU-07			
Título	Visualización del estado de las aulas.		
Prioridad	Alta	**Necesidad**	Esencial
Verificabilidad	Alta	**Estabilidad**	Estable
Descripción	Debe ser posible reconocer al momento al observar la interfaz del sistema si un aula se encuentra libre, reservada para la próxima franja horaria, o en uso por una reserva.		

Tabla 17. RU-07: Visualización del estado de las aulas.

RU-08			
Título	Visualización de la ocupación (número de usuarios) de las aulas.		
Prioridad	Media	**Necesidad**	Deseable
Verificabilidad	Alta	**Estabilidad**	Inestable
Descripción	Debe ser posible reconocer al momento al observar la interfaz del sistema el número de usuarios (traducido como el número de inicios de sesión físicos en los equipos) que se encuentran en un aula.		

Tabla 18. RU-08: Visualización de la ocupación (número de usuarios) de las aulas.

RU-09			
Título	Visualización de la planificación diaria (reservas) de las aulas.		
Prioridad	Alta	**Necesidad**	Esencial
Verificabilidad	Alta	**Estabilidad**	Estable
Descripción	El sistema debe permitir visualizar desde su interfaz, de una forma cómoda y rápida, las reservas de las aulas docentes planificadas para el día actual.		

Tabla 19. RU-09: Visualización de la planificación diaria (reservas) de las aulas.

RU-10			
Título	Visualización del estado de los equipos de las aulas.		
Prioridad	Media	**Necesidad**	Deseable
Verificabilidad	Alta	**Estabilidad**	Inestable
Descripción	El sistema debe permitir visualizar desde su interfaz el estado de los equipos de las aulas de una forma cómoda y rápida. Esto es, para cada equipo, si se encuentra encendido, apagado o en algún estado de error; si está encendido con Windows o Linux; y si está siendo utilizado físicamente por alguien.		

Tabla 20. RU-10: Visualización del estado de los equipos de las aulas.

RU-11			
Título	Disposición de los equipos de las aulas en la interfaz.		
Prioridad	Baja	**Necesidad**	Opcional
Verificabilidad	Alta	**Estabilidad**	Inestable
Descripción	Si se muestran los equipos en la interfaz para cumplir con el requisito anterior, sería preferible que se ordenaran siguiendo la disposición del aula, lo que permitiría identificar fácilmente de forma espacial a cada equipo.		

Tabla 21. RU-11: Disposición de los equipos de las aulas en la interfaz.

RU-12			
Título	Utilizar un monitor externo para mostrar las métricas del CPD.		
Prioridad	Alta	**Necesidad**	Esencial
Verificabilidad	Alta	**Estabilidad**	Estable
Descripción	Al menos uno de los dos monitores externos que proporciona el LDI debe utilizarse para mostrar las métricas obtenidas del CPD y del rack de servidores del laboratorio.		

Tabla 22. RU-12: Utilizar un monitor externo para mostrar las métricas del CPD.

RU-13			
Título	Utilizar un monitor externo adicional para mostrar la información de las aulas.		
Prioridad	Media	**Necesidad**	Deseable
Verificabilidad	Alta	**Estabilidad**	Inestable
Descripción	Por simplificar y separar la información del subsistemas de aulas frente a la información del subsistema del CPD, los técnicos del LDI recomiendan utilizar el otro monitor externo para mostrar solo la información de las aulas, y no sobrecargar el monitor que muestra la información del CPD.		

Tabla 23. RU-13: Utilizar un monitor externo adicional para mostrar la información de las aulas.

RU-14			
Título	Estilo de la GUI del subsistema de aulas.		
Prioridad	Baja	**Necesidad**	Opcional
Verificabilidad	Alta	**Estabilidad**	Inestable
Descripción	Si se utiliza un monitor adicional para mostrar la información relativa al subsistema de aulas, el personal del LDI sugiere que su interfaz de usuario siga un estilo de *«canal de noticias 24h»*. Esto es, un panel principal que ocupe la mayor parte de la pantalla, un panel lateral que presente información tabular, y un panel inferior que vaya mostrando información como si fueran *«titulares de noticias»*, con letras que se desplazan de derecha a izquierda.		

Tabla 24. RU-14: Estilo de la GUI del subsistema de aulas.

RU-15			
Título	Resumen de la información del CPD en el monitor del subsistema de aulas.		
Prioridad	Baja	**Necesidad**	Opcional
Verificabilidad	Alta	**Estabilidad**	Inestable
Descripción	Si se utiliza un monitor adicional para mostrar la información relativa al subsistema de aulas sería recomendable añadir en algún apartado, al que se de poca importancia visual, un resumen de los datos del CPD (temperatura, humedad y SAI).		

Tabla 25. RU-15: Resumen de la información del CPD en el monitor del subsistema de aulas.

RU-16			
Título	No invasivo con la infraestructura actual.		
Prioridad	Media	**Necesidad**	Deseable
Verificabilidad	Media	**Estabilidad**	Estable
Descripción	El sistema desarrollado, y sus componentes o subsistemas, deben evitar en la medida de lo posible ser invasivo con la infraestructura actual. Esto es, evitar que para el desarrollo o integración del sistema sea necesario hacer alguna modificación en la configuración o arquitectura de la infraestructura actual (sin tener en cuenta las adiciones en el panel de Zabbix).		

Tabla 26. RU-16: No invasivo con la infraestructura actual.

RU-17			
Título	Creación de script de instalación.		
Prioridad	Baja	**Necesidad**	Opcional
Verificabilidad	Alta	**Estabilidad**	Inestable
Descripción	El personal del LDI solicita la creación de un script interactivo de instalación del sistema.		

Tabla 27. RU-17: Creación de script de instalación.

Casos de uso

De los requisitos de usuario anteriores se han identificado varios casos de usos que derivan de estos. Los casos de uso se acompañan de un diagrama y se especifican en una tabla que sigue la siguiente platilla:

Identificador	
Título	
Actores	
Objetivo	
Escenario	
Precondiciones	
Postcondiciones	

Tabla 28. Plantilla casos de uso.

Cada uno de los campos de la tabla 28 tiene el siguiente significado:

- **Identificador:** código que identifica unívocamente al caso de uso. Sigue la forma *CU-XX*, donde *CU* son las siglas para *Caso de Uso* y *XX* representa el identificador numérico del caso de uso (01-99).
- **Título:** descripción breve del caso de uso.
- **Actores:** rol de cada usuario que interactúa con el sistema en cada caso de uso.
- **Objetivo:** propósito o intención del usuario al interactuar con el sistema.
- **Escenario:** secuencia de acciones que el usuario debe completar para cumplir su objetivo.
- **Precondiciones:** condiciones previas que deben cumplirse para que el caso de uso pueda llevarse a cabo.
- **Postcondiciones:** estado del sistema tras completarse el caso de uso.

Se han identificado los siguientes casos de uso:

CU-01	
Título	Comprobar ocupación de las aulas para proceder a apertura de nuevos espacios.
Actores	Técnicos del LDI y estudiantes que acuden al despacho solicitando apertura.
Objetivo	Los técnicos del LDI quieren comprobar la ocupación de las aulas del edificio Torres Quevedo antes de proceder o no a la apertura de las aulas del edificio Sabatini a los estudiantes.
Escenario	1. Uno o varios estudiantes acuden al despacho del LDI solicitando la apertura de una aula en el edificio Sabatini para la realización de sus prácticas y necesidades académicas. Alegan para ello que las aulas del edificio Torres Quevedo están llenas o reservadas. 2. Los técnicos del LDI comprueban si realmente las aulas están están llenas u ocupadas con solo visualizar la interfaz del subsistema de aulas en un monitor en la pared del despacho, o abriendo la interfaz del subsistema en un navegador web en su propio equipo. 3. Con la información obtenida, los técnicos valoran proceder o no a la apertura de una aula del edificio Sabatini.
Precondiciones	El subsistema de aulas está funcionando y muestra en su interfaz las reservas de las aulas y la ocupación de las mismas. Esta interfaz se está proyectando en un monitor en la pared del despacho del LDI y/o abierta a la subred del LDI.
Postcondiciones	El estado del sistema no se ve modificado. Los técnicos son capaces de tomar una decisión fundada.

Tabla 29. CU-01: Comprobar ocupación de las aulas para proceder a apertura de nuevos espacios.

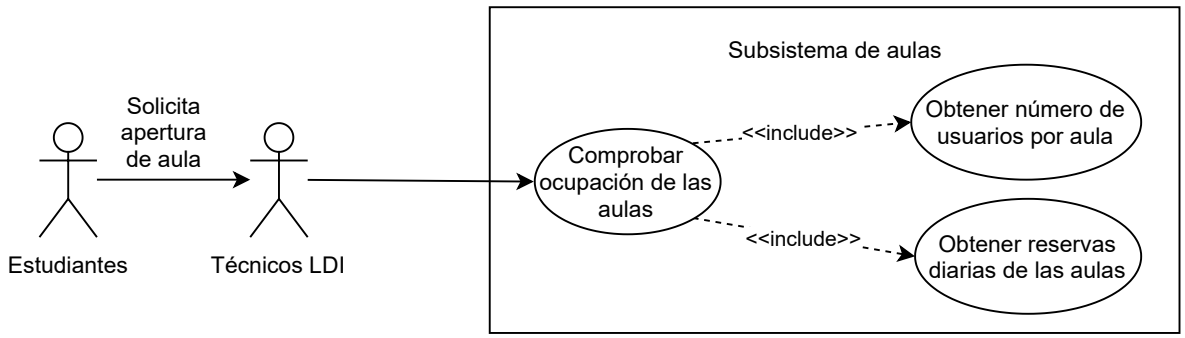

Figura 3: CU-01: Comprobar ocupación de las aulas para proceder a apertura de nuevos espacios.

CU-02	
Título	Comprobar ocupación de las aulas para proceder a realizar mantenimiento no planificado.
Actores	Técnicos del LDI.
Objetivo	Los técnicos del LDI quieren comprobar la ocupación de alguna o varias aulas docentes para proceder o no a la realización de un mantenimiento prioritario no planificado.
Escenario	1. Surge una situación o se conoce un problema con el equipamiento de las aulas que debe ser corregido con cierta prioridad. Para realizar el mantenimiento es preciso que el aula o aulas se encuentren necesariamente libres (sin reserva) y preferiblemente vacías (sin estudiantes). 2. Los técnicos del LDI comprueban si las aulas están ocupadas o en uso con solo visualizar la interfaz del subsistema de aulas en un monitor en la pared del despacho, o abriendo la interfaz del subsistema en un navegador web en su propio equipo. 3. Con la información obtenida, los técnicos valoran proceder o no a realizar el mantenimiento en el aula o aulas deseadas, así como si trasladar a otro aula los estudiantes que pudieran estar utilizando el aula objetivo.
Precondiciones	El subsistema de aulas está funcionando y muestra en su interfaz las reservas de las aulas y la ocupación de las mismas. Esta interfaz se está proyectando en un monitor en la pared del despacho del LDI y/o abierta a la subred del LDI.
Postcondiciones	El estado del sistema no se ve modificado. Los técnicos son capaces de tomar una decisión fundada.

Tabla 30. CU-02: Comprobar ocupación de las aulas para proceder a realizar mantenimiento no planificado.

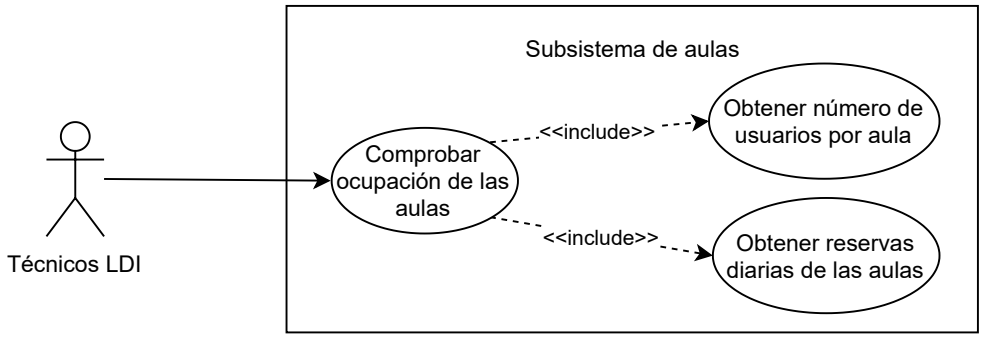

Figura 4: CU-02: Comprobar ocupación de las aulas para proceder a realizar mantenimiento no planificado.

CU-03	
Título	Comprobar temperatura y humedad del CPD.
Actores	Técnicos del LDI.
Objetivo	Los técnicos del LDI quieren comprobar los valores de temperatura y humedad en el interior del CPD para asegurarse que las condiciones son adecuadas para el servicio, y realizar los cambios correspondientes en caso contrario.
Escenario	1. Como parte de las tareas diarias, los técnicos proceden a revisar la situación dentro del CPD. Deben asegurarse que la temperatura y humedad están en valores adecuados, y en caso contrario modificar la refrigeración de la sala. 2. Los técnicos obtienen la información de temperatura y humedad solo con visualizar la interfaz del subsistema del CPD en un monitor en la pared del despacho, o abriendo la interfaz del subsistema en un navegador web en su propio equipo. 3. Con la información obtenida los técnicos comprueban si se dan las correctas condiciones para el funcionamiento del servicio y valoran proceder o no a reajustar la refrigeración de la sala.
Precondiciones	El subsistema del CPD está funcionando y muestra en su interfaz la temperatura y humedad actual del CPD. Esta interfaz se está proyectando en un monitor en la pared del despacho del LDI y/o abierta a la subred del LDI.
Postcondiciones	El estado del sistema no se ve modificado. Los técnicos son capaces de asegurar las condiciones del servicio.

Tabla 31. CU-03: Comprobar temperatura y humedad del CPD.

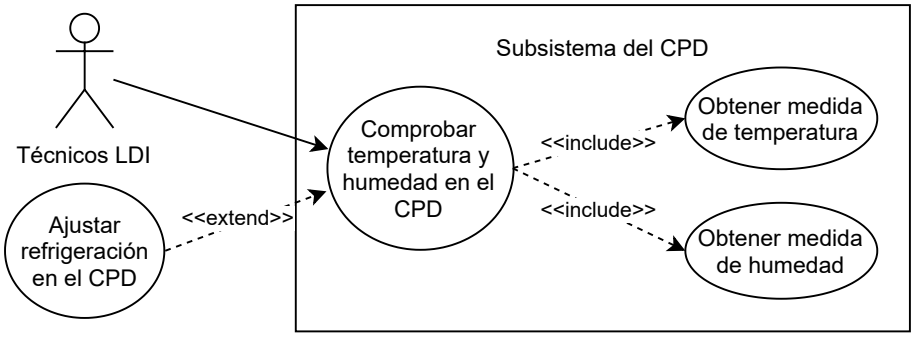

Figura 5: CU-03: Comprobar temperatura y humedad del CPD.

CU-04	
Título	Situación de alarma en el CPD.
Actores	Técnicos del LDI.
Objetivo	El sistema detecta un problema o una situación anómala en el CPD y lanza una alarma para indicar a los técnicos del LDI que deben ocuparse de la misma.
Escenario	1. El subsistema del CPD se encuentra continuamente monitorizando los valores de temperatura y humedad en la sala, así como el estado y nivel de carga del SAI del rack del LDI. 2. El subsistema detecta una situación problemática (p.e. temperatura elevada en la sala, o rack desconectado de la alimentación eléctrica y nivel de carga de las baterías del SAI descendiendo). 3. El subsistema lanza una alerta visual y sonora en el despacho del LDI, donde se encuentran los técnicos. 4. Los técnicos comprueban la interfaz del subsistema del CPD para conocer la emergencia y valorar como proceder.
Precondiciones	El subsistema del CPD está funcionando y monitorizando continuamente el estado del CPD y el rack del LDI. El subsistema detecta un problema o una situación anómala en el CPD.
Postcondiciones	El susbsistema lanza una alarma en el despacho. Los técnicos analizan la situación y valoran cómo proceder.

Tabla 32. CU-04: Situación de alarma en el CPD.

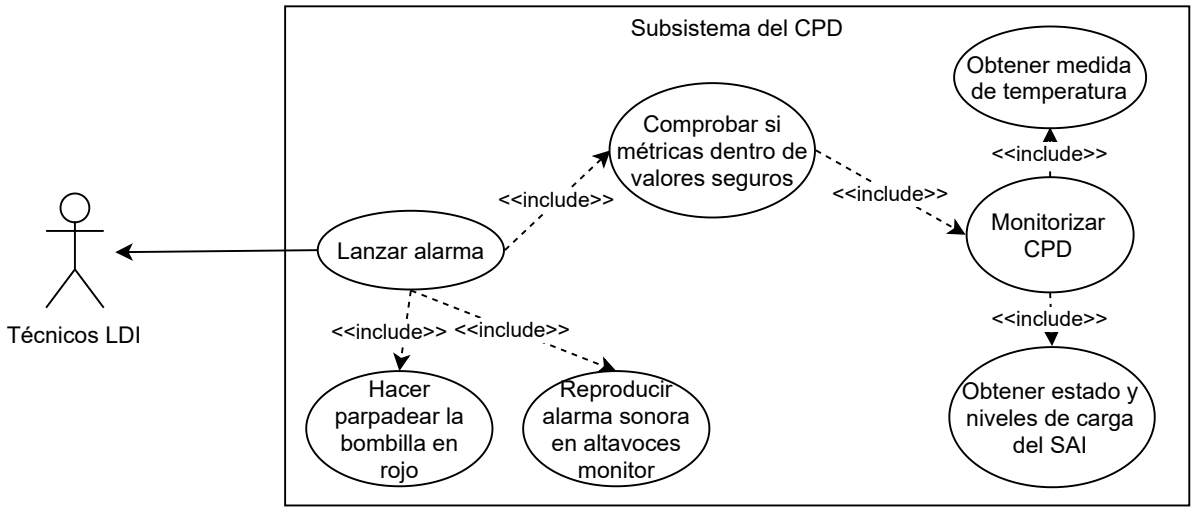

Figura 6: CU-04: Situación de alarma en el CPD.

CU-05	
Título	Aviso de CPD en uso por el personal del Departamento.
Actores	Técnicos del LDI y docentes e investigadores (PDI) del Dpto. de Informática.
Objetivo	PDI del Dpto. de Informática se encuentra dentro del CPD. Los técnicos deben abandonar el despacho para alguna tarea de mantenimiento en las aulas y cerrar con llave.
Escenario	1. PDI del Dpto. de Informática se encuentra físicamente dentro del CPD realizando alguna labor. 2. Los técnicos del LDI deben abandonar el despacho y cerrar con llave. De camino a la puerta del mismo se encuentran con la bombilla Wi-Fi encendida que alerta de la presencia de alguien en el CPD. 3. Los técnicos han sido avisados y deben esperar a que el PDI termine sus labores antes de abandonar el despacho o bien debe quedarse al menos uno de los técnicos en el mismo, pues si cerraran el despacho dejarían encerrado al PDI.
Precondiciones	El subsistema del CPD está funcionando y monitorizando continuamente la iluminación de la sala. En la pared del despacho del LDI se encuentra la bombilla Wi-Fi que refleja el estado de iluminación del CPD.
Postcondiciones	Si la luz del CPD continúa encendida, la bombilla del despacho permanecerá encendida. Los técnicos analizan la situación y valoran como proceder.

Tabla 33. CU-05: Aviso de CPD en uso por el personal del Departamento.

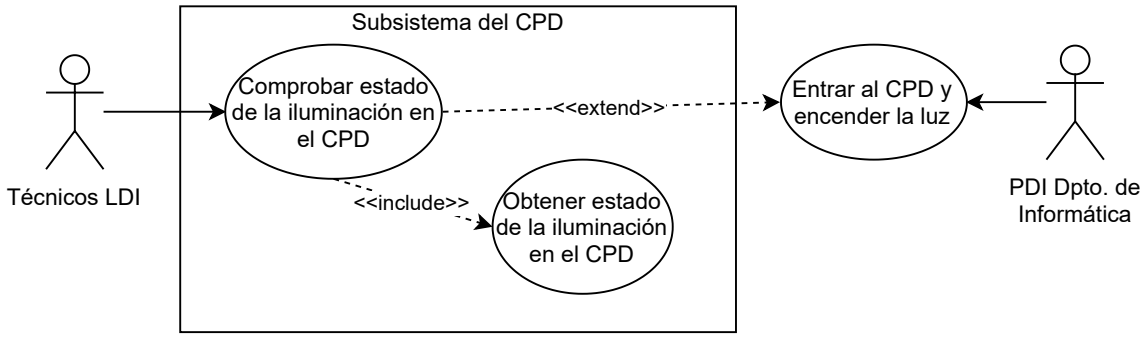

Figura 7: CU-05: Aviso de CPD en uso por el personal del Departamento.

Requisitos software

De los requisitos de usuario definidos con anterioridad, se han identificado e implementado los siguientes requisitos software **finales**. Los requisitos software **finalmente implementados** se presentan en forma tabular siguiendo la siguiente plantilla.

Respecto de la plantilla de los RU se han eliminado los apartados *necesidad* y *estabilidad*, pues carecen de sentido para indicar un requisito ya implementado. En su lugar se han sustituido por el apartado *procedencia*, que se explica a continuación.

Identificador					
Título					
Prioridad		**Verificabilidad**		**Procedencia**	
Descripción					

Tabla 34. Plantilla requisitos software.

Cada uno de los campos de la tabla 34 puede tomar los siguientes valores:

- **Identificador:** código que identifica unívocamente al requisito. Será de la forma *[RF,RNF]-XX*, donde *RF* o *RNF* son las siglas para *Requisito Funcional* y *Requisito No Funcional* respectivamente, mientras que *XX* representa el identificador numérico del requisito (01-99).
- **Título:** descripción breve del requisito.
- **Prioridad:** indica cómo de prioritaria ha sido la implementación del requisito respecto a los demás. Puede tomar los valores *alta*, *media* y *baja*.
- **Verificabilidad:** indica en qué grado es posible comprobar si el requisito software se ha implementado correctamente. Puede tomar los valores *alta*, *media* y *baja*.
- **Procedencia:** identificador del requisito de usuario del cual se ha derivado el actual requisito software.
- **Descripción:** descripción de las especificaciones del requisito software.

Se procede a indicar los requisitos software **finalmente implementados**.

Requisitos funcionales

RF-01					
Título	Monitorización de la temperatura actual del CPD.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-01
Descripción	El sistema monitorizará la temperatura de la sala del CPD cada 60 segundos.				

Tabla 35. RF-01: Monitorización de la temperatura actual del CPD.

RF-02					
Título	Visualización de la temperatura actual del CPD en la interfaz del sistema.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-01
Descripción	El panel de Grafana contendrá un elemento de tipo *Gauge* (velocímetro) que mostrará la temperatura actual del CPD. Se dividirá en tres secciones de diferentes colores: 10-29.9ºC (verde), 30-34.9ºC (naranja) y 35-50ºC (rojo).				

Tabla 36. RF-02: Visualización de la temperatura actual del CPD en la interfaz del sistema.

RF-03					
Título	Monitorización de la humedad actual del CPD.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-02
Descripción	El sistema monitorizará la humedad de la sala del CPD cada 60 segundos.				

Tabla 37. RF-03: Monitorización de la humedad actual del CPD.

RF-04					
Título	Visualización de la humedad actual del CPD en la interfaz del sistema.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-02
Descripción	El panel de Grafana contendrá un elemento de tipo *Gauge* (velocímetro) que mostrará la humedad actual del CPD. Se dividirá en tres secciones de diferentes colores: 0-19.9 % (rojo), 20-80 % (verde) y 80.1-100 % (rojo).				

Tabla 38. RF-04: Visualización de la humedad actual del CPD en la interfaz del sistema.

RF-05					
Título	Obtención de los datos de nivel de carga del SAI.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-03, RU-06
Descripción	El sistema obtendrá los datos del nivel de carga del SAI desde Zabbix en el servidor ultraheroe, donde se encuentran antes de comenzar a desarrollar el sistema.				

Tabla 39. RF-05: Obtención de los datos de nivel de carga del SAI.

RF-06					
Título	Visualización del nivel de carga actual del SAI en la interfaz del sistema.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-03
Descripción	El panel de Grafana contendrá un elemento de tipo *Gauge* (velocímetro) que mostrará el nivel de carga actual del SAI. Se dividirá en tres secciones de diferentes colores: 0-20 % (rojo), 20.1-50 % (naranja) y 50.1-100 % (verde).				

Tabla 40. RF-06: Visualización del nivel de carga actual del SAI en la interfaz del sistema.

RF-07					
Título	Monitorización del estado actual de la iluminación del CPD.				
Prioridad	Media	**Verificabilidad**	Alta	**Procedencia**	RU-04
Descripción	El sistema monitorizará el estado actual de la iluminación de la sala del CPD cada 10 segundos.				

Tabla 41. RF-07: Monitorización del estado actual de la iluminación del CPD.

RF-08					
Título	Visualización del estado actual de la iluminación del CPD.				
Prioridad	Media	**Verificabilidad**	Alta	**Procedencia**	RU-04
Descripción	La visualización del estado actual de la iluminación del CPD se realizará mediante la bombilla Wi-Fi Philips® que se situará junto a la puerta de entrada al CPD. La bombilla permanecerá encendida con un 20 % de brillo si la luz dentro del CPD está encendida, y permanecerá apagada en caso contrario.				

Tabla 42. RF-08: Visualización del estado actual de la iluminación del CPD.

RF-09					
Título	Alerta visual de temperatura elevada en el CPD.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-05
Descripción	La bombilla Wi-Fi Philips® utilizada en el requisito RF-08 parpadeará en rojo si la temperatura dentro del CPD se eleva por encima de 30ºC. Este estado de parpadeo tendrá prioridad sobre cualquier modificación del estado de la bombilla definido en el requisito RF-08.				

Tabla 43. RF-09: Alerta visual de temperatura elevada en el CPD.

RF-10					
Título	Alerta sonora de temperatura elevada en el CPD.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-05
Descripción	El monitor que muestre los valores y métricas recogidos para el CPD emitirá una alerta sonora (reproducirá un sonido de sirena) durante 8 segundos si la temperatura dentro del CPD se eleva por encima de 30ºC.				

Tabla 44. RF-10: Alerta sonora de temperatura elevada en el CPD.

RF-11					
Título	Alerta visual de desconexión de la red eléctrica del rack del LDI.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-05
Descripción	La bombilla Wi-Fi Philips® definida en el requisito RF-08 parpadeará en rojo si el rack del LDI sufre un corte del suministro de la red eléctrica y por lo tanto el SAI entra en modo batería. Este estado de parpadeo tendrá prioridad sobre cualquier modificación del estado de la bombilla definido en el requisito RF-08.				

Tabla 45. RF-11: Alerta visual de desconexión de la red eléctrica del rack del LDI.

RF-12					
Título	Alerta sonora de desconexión de la red eléctrica del rack del LDI.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-05
Descripción	El monitor que muestre los valores y métricas recogidos del CPD emitirá una alerta sonora (reproducirá un sonido de sirena) durante 8 segundos si el rack del LDI sufre un corte del suministro de la red eléctrica y por lo tanto el SAI entra en modo batería.				

Tabla 46. RF-12: Alerta sonora de desconexión de la red eléctrica del rack del LDI.

RF-13					
Título	Obtención de los datos de reservas de las aulas.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-07, RU-09, RU-16
Descripción	La planificación de las reservas de las aulas para el día de hoy se obtendrá desde el apartado correspondiente la web del LDI [10]. De esta forma, se consigue que el nuevo sistema no sea invasivo con la infraestructura actual (RU-16). Para parsear la web se desarrollará un sencillo *crawler* o araña web que se integrará en el sistema.				

Tabla 47. RF-13: Obtención de los datos de reservas de las aulas.

RF-14					
Título	Diseño de la interfaz de usuario del subsistema de aulas.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-14
Descripción	La interfaz de usuario del subsistema de aulas se dividirá en cuatro paneles: • Un panel superior que ocupe todo el ancho del monitor y un 10 % de su alto, que incluirá el nombre del LDI y un reloj con día de la semana, día del mes, mes, hora y minutos. • Un panel inferior que ocupe todo el ancho del monitor y un 10 % de su alto. Su función y uso quedarán definidos en posteriores requisitos. • Un panel principal a la izquierda, que ocupará un 80 % del alto del monitor y un 70 % de su ancho. Se compondrá de diversas pantallas o vistas que irán rotando cada 10 segundos. Su función y uso quedarán definidos en posteriores requisitos. • Un panel secundario a la derecha, que ocupará un 80 % del alto del monitor y un 30 % de su ancho. Su función y uso quedarán definidos en posteriores requisitos.				

Tabla 48. RF-14: Diseño de la interfaz de usuario del subsistema de aulas.

RF-15					
Título	Resumen de la información del CPD en la interfaz del subsistema de aulas.				
Prioridad	Baja	**Verificabilidad**	Alta	**Procedencia**	RU-15
Descripción	En el panel inferior del subsistema de aulas se mostrará la información obtenida para el CPD de forma resumida (temperatura, humedad y estado del SAI) mediante texto que se desplaza de derecha a izquierda.				

Tabla 49. RF-15: Resumen de la información del CPD en la interfaz del subsistema de aulas.

RF-16					
Título	Visualización del estado de las aulas.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-07
Descripción	En el panel secundario del subsistema de aulas se mostrará una disposición tabular de las aulas en 4 filas y una columna. Cada celda (fila) representará un aula y se indicará la misma en su interior. La celda tomará un color acorde al estado del aula: • Verde: el aula se encuentra libre para que cualquier estudiante la utilice o los técnicos realicen mantenimiento. • Amarillo: el aula tiene una reserva que empezará en 30 minutos o menos. • Parpadeando amarillo-naranja: el aula tiene una reserva que empezará en 10 minutos o menos. • Rojo: el aula está ocupada por una reserva.				

Tabla 50. RF-16: Visualización del estado de las aulas.

RF-17					
Título	Visualización de la ocupación (número de usuarios) de las aulas.				
Prioridad	Media	**Verificabilidad**	Alta	**Procedencia**	RU-08
Descripción	En la disposición tabular definida en el requisito anterior se incluirá en cada celda un indicador numérico que representa el número de inicios de sesión físicos en el aula. Para obtener estos valores se hará uso desde el servidor *instalador* del script `comprobar_ocupacion.py` definido en el TFG de Lázaro [11].				

Tabla 51. RF-17: Visualización de la ocupación (número de usuarios) de las aulas.

RF-18					
Título	Visualización de la planificación diaria (reservas) de las aulas.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-09
Descripción	En el panel principal del subsistema de aulas se mostrará un máximo de ocho tarjetas que se corresponderán con las próximas ocho reservas para el día de hoy. Se mostrarán agrupadas en un máximo de cuatro tarjetas formando así dos pantallas rotativas. En las tarjetas se mostrará tanta información como se haya recuperado conforme al RF-13. Si no hubiera reservas para el día de hoy o ya hubieran finalizado estas, se indicará mediante un mensaje en el panel.				

Tabla 52. RF-18: Visualización de la planificación diaria (reservas) de las aulas.

RF-19					
Título	Visualización del estado de los equipos de las aulas.				
Prioridad	Media	**Verificabilidad**	Alta	**Procedencia**	RU-10
Descripción	En el panel principal del subsistema de aulas se mostrará una pantalla rotativa por cada aula. Se mostrará para cada aula un conjunto de tarjetas igual al número de equipos que posee esta. Cada tarjeta (equipo) deberá incluir la siguiente información: • Nombre de host del equipo. • Si el equipo se encuentra apagado, encendido, o en estado de error (imposible conectar). • En caso de que el equipo se encuentre encendido, si está ejecutando Windows o GNU/Linux. • En caso de que el equipo se encuentre encendido, si está siendo utilizado físicamente por alguien. Esta información se obtendrá mediante la ejecución desde el servidor *instalador* del script `comprobar_ocupacion.py` definido en el TFG de Lázaro [11].				

Tabla 53. RF-19: Visualización del estado de los equipos de las aulas.

RF-20					
Título	Disposición de los equipos de las aulas en la interfaz.				
Prioridad	Baja	**Verificabilidad**	Alta	**Procedencia**	RU-11
Descripción	Las tarjetas que representan los equipos en el requisito anterior se ordenarán siguiendo la disposición espacial del aula que representan.				

Tabla 54. RF-20: Disposición de los equipos de las aulas en la interfaz.

RF-21					
Título	Identificación del aula que se muestra en el panel principal.				
Prioridad	Media	**Verificabilidad**	Alta	**Procedencia**	RU-10
Descripción	Se aprovechará el panel secundario de la interfaz, donde aparecen los indicadores de cada aula, para identificar qué aula se está mostrando en cada momento en el panel principal de la interfaz del subsistema de aulas. El aula en el panel secundario correspondiente al aula activa en el panel principal incluirá el indicativo de una flecha. Asimismo, el resto de aulas del panel secundario reducirán su ancho un 20 %.				

Tabla 55. RF-21: Identificación del aula que se muestra en el panel principal.

Requisitos no funcionales

RNF-01					
Título	Utilización del módulo de sensores SenseHat para las métricas del CPD.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	-
Descripción	Se utilizará un módulo de sensores SenseHat [3] conectado a uno de los minicomputadores Raspberry Pi para medir la temperatura y humedad relativa dentro del CPD. Este minicomputador Raspberry Pi se situará dentro del CPD y será definido como *rpi1*.				

Tabla 56. RNF-01: Utilización del módulo de sensores SenseHat para las métricas del CPD.

RNF-02					
Título	Envío de los datos de temperatura y humedad del CPD a Zabbix.				
Prioridad	Media	**Verificabilidad**	Alta	**Procedencia**	RU-06
Descripción	El sistema creará en el directorio `/tmp` de *rpi1* dos ficheros que contendrán respectivamente el último valor numérico de la temperatura y la humedad leídos por el sistema. El agente de zabbix (`zabbix_agentd`) se configurará para leer las métricas de estos ficheros cada 60 segundos.				

Tabla 57. RNF-02: Envío de los datos de temperatura y humedad del CPD a Zabbix.

RNF-03					
Título	Visualización de la temperatura actual del CPD en el panel led del módulo SenseHat.				
Prioridad	Baja	**Verificabilidad**	Alta	**Procedencia**	RU-01
Descripción	Se deberá aprovechar el panel led del módulo SenseHat [3] para mostrar el valor la temperatura actual del CPD redondeado al entero más cercano.				

Tabla 58. RNF-03: Visualización de la temperatura actual del CPD en el panel led del módulo SenseHat.

RNF-04					
Título	Utilización del módulo de cámara para monitorizar la iluminación del CPD.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	-
Descripción	Se utilizará un módulo de cámara conectado a uno de los minicomputadores Raspberry Pi para monitorizar el estado de la iluminación del CPD. Este minicomputador Raspberry Pi deberá pues ser colocado dentro del CPD, y será definido como *rpi1*.				

Tabla 59. RNF-04: Utilización del módulo de cámara para monitorizar la iluminación del CPD.

RNF-05					
Título	Utilización de un monitor para la interfaz del subsistema del CPD.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-12
Descripción	Se utilizará un monitor para reproducir la instancia de Grafana que muestra las métricas del subsistema del CPD. Este monitor se situará dentro del despacho del LDI (4.0.F11) sobre la puerta del CPD. Se conectará mediante HDMI a uno de los minicomputadores Raspberry Pi que será definido como *rpi2*.				

Tabla 60. RNF-05: Utilización de un monitor para la interfaz del subsistema del CPD.

RNF-06					
Título	Utilización de un monitor para la interfaz del subsistema de aulas.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-13
Descripción	Se utilizará un monitor para reproducir la interfaz de usuario del subsistema de aulas. Este monitor se situará dentro del despacho del LDI (4.0.F11) y entre las dos puertas de entrada al mismo. Se conectará mediante HDMI a uno de los minicomputadores Raspberry Pi que será definido como *rpi3*.				

Tabla 61. RNF-06: Utilización de un monitor para la interfaz del subsistema de aulas.

RNF-07					
Título	Comunicación entre subsistemas mediante API REST.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	RU-10
Descripción	La comunicación entre los distintos subsistemas se realizará mediante peticiones HTTP a dos API REST desarrolladas: • **rpi2_api:** punto centralizado de control del subsistema del CPD. Debe comprobar si las métricas enviadas por *rpi1* acerca del CPD son adecuadas. Debe controlar las alarmas visuales y sonoras. Debe controlar el indicativo de iluminación del CPD. Debe proporcionar las lecturas obtenidas por rpi1 del CPD mediante peticiones GET. Se ejecuta en *rpi2*. • **rpi3_api:** punto centralizado de control del subsistema de aulas. Se encargará de recoger, parsear y formatear la información de la web [10] y los scripts de Lázaro [11]. Debe proporcionar información sobre las reservas y estados de las aulas así como la ocupación de las mismas mediante peticiones GET. Se ejecuta en *rpi3*. La arquitectura y definición de las API es abordada en el capítulo de diseño.				

Tabla 62. RNF-07: Comunicación entre subsistemas mediante API REST.

RNF-08					
Título	Autenticación en las peticiones de escritura a las API				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	-
Descripción	Todas las peticiones de escritura o actualización (POST, PUT) realizadas a las API REST definidas deberán estar autenticadas mediante un *Bearer token* secreto y común a todas ellas.				

Tabla 63. RNF-08: Autenticación en las peticiones de escritura a las API.

RNF-09					
Título	Protección por cortafuegos de las API				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	-
Descripción	Todas las API deben estar protegidas por un cortafuegos y restringidas a las subredes del laboratorio: 163.117.142.0/24 y 163.117.170.0/24.				

Tabla 64. RNF-09: Protección por cortafuegos de las API.

RNF-10					
Título	Protección por cortafuegos de las interfaces de usuario.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	-
Descripción	Todas las interfaces de usuario (Grafana rpi2 y Gatsby rpi3) deben estar protegidas por un cortafuegos y restringidas a las subredes de la Universidad Carlos III de Madrid: 163.117.0.0/16.				

Tabla 65. RNF-10: Protección por cortafuegos de las interfaces de usuario.

RNF-11					
Título	Compatibilidad con la arquitectura de procesador ARM.				
Prioridad	Alta	**Verificabilidad**	Alta	**Procedencia**	-
Descripción	Dado que la infraestructura principal del sistema se apoya en minicomputadores Raspberry Pi, que posees arquitectura de procesador ARM, todo los binarios desarrollados deben ser compatibles con esta arquitectura.				

Tabla 66. RNF-11: Compatibilidad con la arquitectura de procesador ARM.

RNF-12					
Título	Optimización del sistema.				
Prioridad	Media	**Verificabilidad**	Media	**Procedencia**	-
Descripción	Dado que la infraestructura principal del sistema se apoya en minicomputadores Raspberry Pi, con una potencia reducida, el sistema debe estar optimizado para requerir los menos recursos posibles de procesador y memoria.				

Tabla 67. RNF-12: Optimización del sistema.

RNF-13					
Título	Ahorro de energía de los monitores.				
Prioridad	Baja	**Verificabilidad**	Alta	**Procedencia**	-
Descripción	Para ahorrar energía eléctrica y alargar la vida útil de los monitores estos deberán encenderse automáticamente de lunes a viernes a las 8:30 y apagarse automáticamente a las 21:30.				

Tabla 68. RNF-13: Ahorro de energía de los monitores.

RNF-14					
Título	Creación de script de instalación.				
Prioridad	Baja	**Verificabilidad**	Alta	**Procedencia**	RU-17
Descripción	Se debe definir un script de instalación interactivo, que pregunte al usuario qué subsistema desea instalar/desplegar (rpi1, rpi2, rpi3). El script sugerirá durante la instalación los ajustes (p.e. dirección IP, hostname) por defecto necesarios para el correcto funcionamiento del sistema y ofrecerá la posibilidad al usuario de modificarlos.				

Tabla 69. RNF-14: Creación de script de instalación.

RNF-15					
Título	Protección ante fallos de red y pérdidas de conexión.				
Prioridad	Media	**Verificabilidad**	Alta	**Procedencia**	-
Descripción	El sistema deberá ser resistente ante fallos de red a la hora de obtener la información. Para ello, deberá informar al usuario cuando ha perdido la conexión y los datos no se pueden actualizar. Asimismo, si la conexión vuelve, el sistema deberá recuperar su completa funcionalidad automáticamente.				

Tabla 70. RNF-15: Protección ante fallos de red y pérdidas de conexión.

Matriz de trazabilidad entre requisitos de usuario y requisitos de software

Como proceso de verificación de los requisitos del sistema se ha definido en la tabla 71 una matriz de trazabilidad para cerciorar que **todos los requisitos del usuario quedan cubiertos por al menos uno de los requisitos software** definidos anteriormente.

Como se puede observar en la matriz de trazabilidad de la tabla 71, como al menos cada requisito de usuario (columnas) está cubierto por un requisito de software (hay al menos una «X» por columna), **podemos verificar y afirmar que se han cubierto todas las necesidades del usuario** con los requisitos software identificados.

En la matriz de trazabilidad hay requisitos software que no cubren ningún requisito de usuario (filas sin ninguna «X»). Esto es debido a que son requisitos no funcionales que han surgido como restricciones de diseño, o bien son medidas habituales como resistencia a errores o seguridad del software que habitualmente los usuarios no exigen explícitamente pero se dan por supuesto en un trabajo de ingeniería.

	RU-01	RU-02	RU-03	RU-04	RU-05	RU-06	RU-07	RU-08	RU-09	RU-10	RU-11	RU-12	RU-13	RU-14	RU-15	RU-16	RU-17
RF-01	X																
RF-02	X																
RF-03		X															
RF-04		X															
RF-05			X			X											
RF-06			X														
RF-07				X													
RF-08				X													
RF-09					X												
RF-10					X												
RF-11					X												
RF-12					X												
RF-13							X		X							X	
RF-14														X			
RF-15															X		
RF-16							X										
RF-17								X									
RF-18									X								
RF-19										X							

	RU-01	RU-02	RU-03	RU-04	RU-05	RU-06	RU-07	RU-08	RU-09	RU-10	RU-11	RU-12	RU-13	RU-14	RU-15	RU-16	RU-17
RF-20											X						
RF-21										X							
RNF-01																	
RNF-02						X											
RNF-03	X																
RNF-04																	
RNF-05												X					
RNF-06													X				
RNF-07										X							
RNF-08																	
RNF-09																	
RNF-10																	
RNF-11																	
RNF-12																	
RNF-13																	
RNF-14																	X
RNF-15																	

Tabla 71. Matriz de trazabilidad entre requisitos de usuario y requisitos de software.

Diseño

El sistema a desarrollar estará compuesto por **dos subsistemas principales**: el subsistema del **CPD** y el subsistema de **aulas**. El subsistema del CPD a su vez estará compuesto por los minicomputadores *rpi1* y *rpi2*, los módulos de sensores SenseHat y cámara, la bombilla Wi-Fi y un monitor. Este subsistema hará uso del servidor ultraheroe de la infraestructura heredada. El subsistema de aulas, por su parte, solo se compondrá de un minicomputador (*rpi3*) y un monitor. Requerirá el uso de los servidores instalador y web de la infraestructura heredada.

Definición de la arquitectura del sistema

La arquitectura propuesta del sistema, que se corresponde con la que ha sido implantada, se muestra en la figura 8.

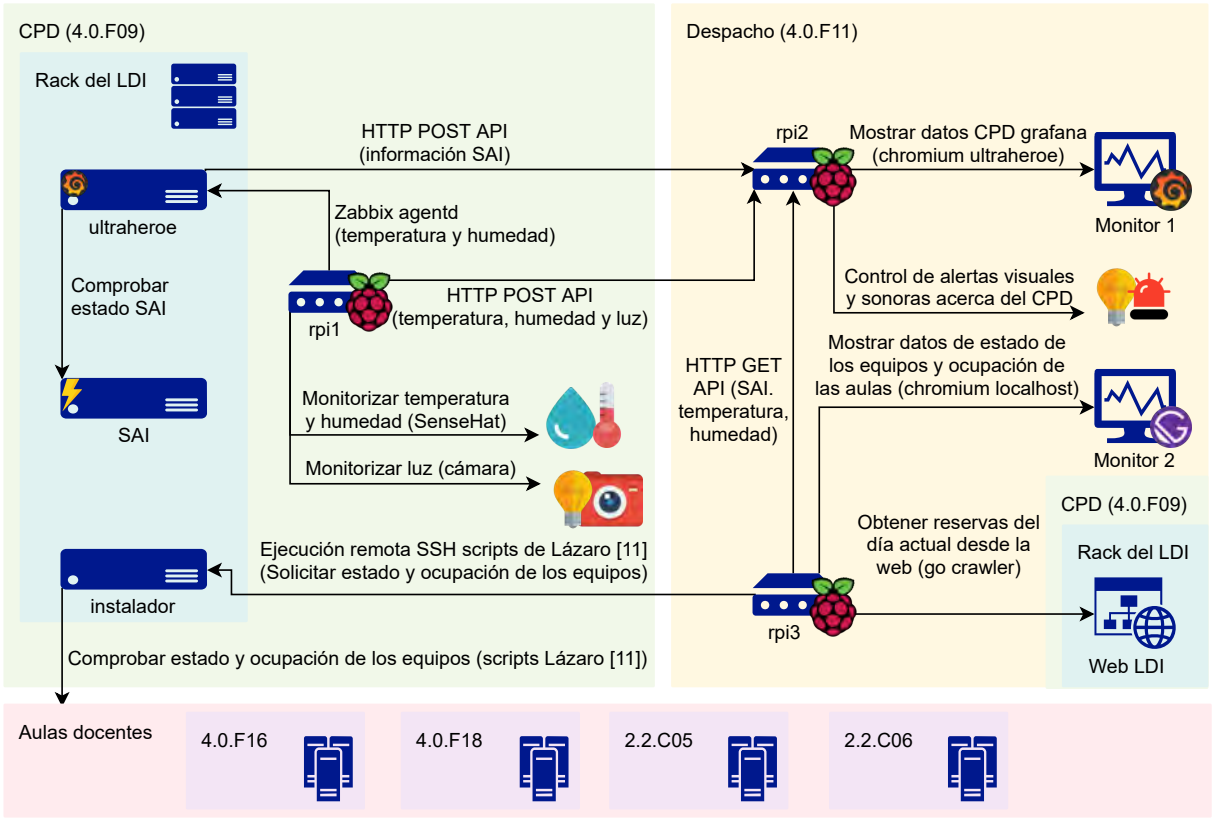

Figura 8: Arquitectura del sistema. Elaboración propia.

Así pues, tenemos **tres escenarios** en el alcance y aplicación del sistema: el **CPD** señalado en verde en la figura 8, el **despacho** señalado en amarillo, y las **aulas** señaladas en tonos malvas. El minicomputador rpi1 se encontrará dentro del CPD para monitorizar la sala con ayuda de los módulos SenseHat y la cámara. Rpi2 y rpi3 se encontrarán en el despacho, cada una conectada a uno de los dos monitores de las paredes. Más adelante en este capítulo se explicará en detalle la arquitectura y funcionalidades de cada minicomputador Raspberry Pi.

En la figura 9 se muestra un esquema textual a modo de resumen de las responsabilidades de cada uno de estos elementos.

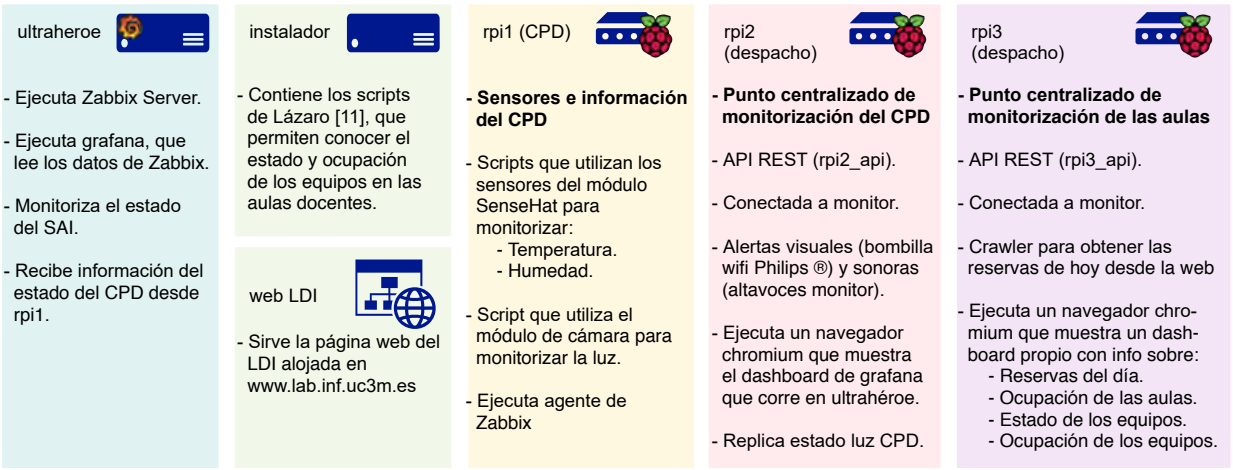

Figura 9: Responsabilidades de la infraestructura que compone el sistema. Elaboración propia.

Infraestructura heredada

En primer lugar se explica brevemente la infraestructura heredada. El rack del LDI dispone, entre otros elementos, de un sistema de alimentación ininterrumpida (SAI) y tres servidores: ultraheroe, instalador y web.

- **Ultraheroe** es utilizado principalmente como un servidor de respaldo. Contiene copias de seguridad y las herramientas necesarias para replicar la infraestructura en caso de catástrofe. Asimismo, es utilizado como entorno de producción para aplicaciones no críticas. En lo que respecta a la arquitectura descrita se encargará de tres tareas principales: monitorizar el estado del **SAI** para informar a rpi2, ejecutar la instancia de servidor de **Zabbix**, y ejecutar la instancia de **Grafana** utilizada para generar el dashboard del subsistema del CPD.
- **Instalador** se utiliza como punto centralizado de gestión. Desde este servidor se comprueba el estado de la infraestructura, se lanzan tareas en remoto, y en definitiva se utiliza para las labores de control y mantenimiento diario de la infraestructura del LDI. Para ello, se utilizan los **scripts** en python desarrollados por **Lázaro** [11]. Rpi3 ejecutará remotamente por SSH en el servidor el script `comprobar_ocupacion.py` y parseará la salida para obtener información acerca del estado y ocupación de los equipos.
- **Web** sirve la página **web** del LDI alojada en `https://www.lab.inf.uc3m.es` con ayuda de un servidor Apache. Desde la misma se puede comprobar la ocupación diaria de las aulas en un apartado específico que muestra las **reservas** de los laboratorios para días seleccionados. Esta información se muestra en forma de tabla pensada para su interpretación por seres humanos. Rpi3 obtendrá las reservas para el día actual a través de la web del LDI, parseando la misma gracias a un crawler incorporado en su API REST (rpi3_api).

Arquitectura de componentes de rpi1

A continuación se procede a analizar en más detalle cada uno de los minicomputadores Raspberry Pi, sus funciones y componentes. En primer lugar se presenta la arquitectura de componentes de rpi1 en la figura 10.

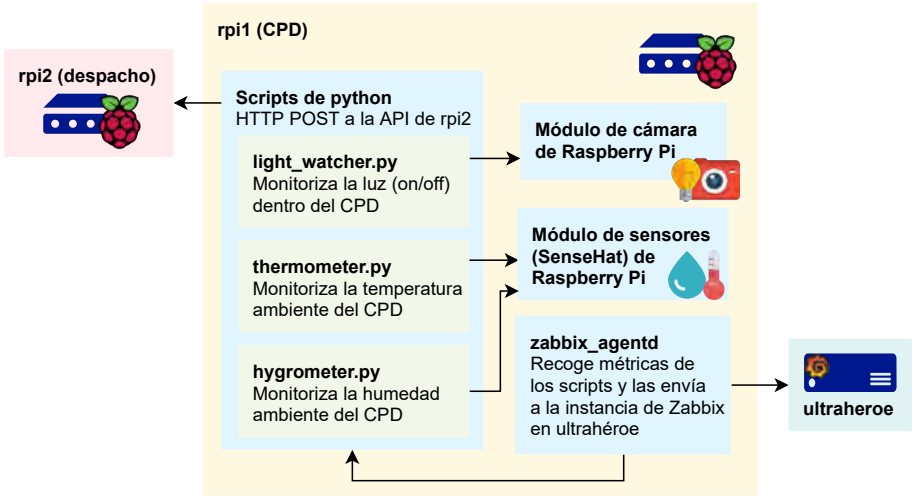

Figura 10: Arquitectura de componentes de la Raspberry Pi 1 (rpi1). Elaboración propia.

Rpi1 se situará físicamente dentro del CPD. Su principal función será la de **analizar el entorno del CPD** y enviar la información recogida a **Zabbix** en ultraheroe y a la API REST de rpi2 (rpi2_api).

En rpi1 se ejecutará el script `light_watcher.py`, que cada 10 segundos tomará una foto con el módulo de la **cámara** y evaluará el brillo de los píxeles para determinar si la **luz dentro del CPD** está encendida o no. Rpi1 también ejecutará los scripts `thermometer.py` e `hygrometer.py` que monitorizarán cada 60 segundos los valores ambientales de **temperatura y humedad** respectivamente con ayuda del módulo de sensores **SenseHat**.

Los valores obtenidos por estos scripts **se enviarán a rpi2** mediante una petición HTTP POST autenticada a la API REST que la misma estará ejecutando. Se analizará el diseño de las distintas API REST y sus peticiones más adelante en este capítulo. A su vez, los scripts `thermometer.py` e `hygrometer.py` depositarán el valor de la última medición realizada en los ficheros de texto `last_temp.txt` y `last_hum.txt` respectivamente, situados en el directorio `/tmp/`.

Zabbix recogerá únicamente las métricas de temperatura y humedad. Esto lo hará mediante un **agente** (`zabbix_agentd`) desarrollado por la comunidad de Zabbix que se ejecutará en rpi1 como un demonio. El agente se configurará con dos parámetros de usuario [35] para que extraiga cada 60 segundos mediante una instrucción `cat` de *bash* los valores de **temperatura y humedad** que se encuentran en los ficheros `last_temp.txt` y `last_hum.txt` mencionados anteriormente.

Arquitectura de componentes de rpi2

En la figura 11 se puede apreciar la arquitectura de componentes de rpi2.

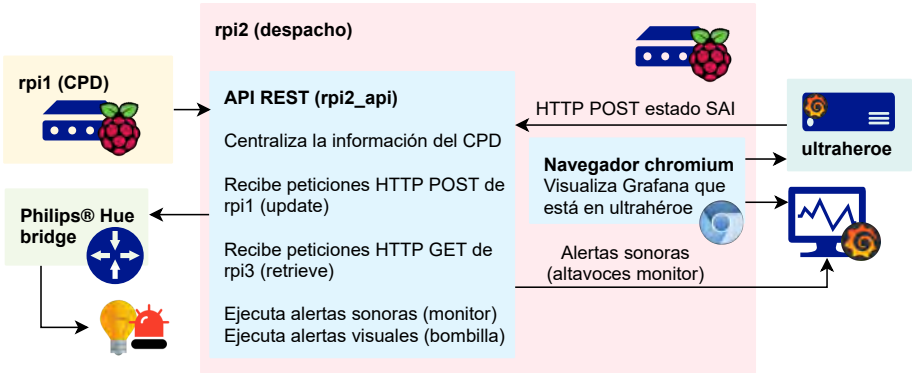

Figura 11: Arquitectura de componentes de la Raspberry Pi 2 (rpi2). Elaboración propia.

Rpi2 ejecutará por una parte un navegador chromium, el cual se cargará al inicio con un **dashboard de Grafana** que mostrará los datos del CPD recogidos por rpi1. La instancia de Grafana se ejecutará en ultraheroe. La proyección se realizará en el monitor conectado a rpi2, que se situará dentro del despacho del LDI (4.0.F11) sobre la puerta que da acceso al CPD.

El principal componente de rpi2 será su **API REST (rpi2_api)**, la cual se detalla más adelante en este capítulo. Rpi2_api servirá como **punto central para coordinar la monitorización del CPD**. Recibirá y almacenará en memoria principal (ya que son datos con una pronta caducidad) los valores de temperatura, humedad e iluminación desde rpi1; y el estado y nivel de carga del SAI desde ultraheroe. Las actualizaciones de estos datos se realizan mediante peticiones HTTP POST autenticadas con un *Bearer token* secreto y común a todas ellas. Asimismo, estos datos estarán disponibles para su consulta externa mediante peticiones HTTP GET sin autenticar.

Rpi2_api estará conectada a la **bombilla Wi-Fi Philips®** mediante su correspondiente Philips® Hue bridge. La integración se realizará mediante el uso de la interfaz *GoHue* [36] con licencia MIT. Rpi2_api mantendrá esta bombilla encendida si la iluminación del CPD está encendida, y apagada en caso contrario.

La propia rpi2_api, al ser actualizados sus valores, se encargará de analizar si estos se encuentran en rangos seguros y lanzará una **alerta en el despacho** en caso contrario. Esta alerta será **sonora y visual**. La alerta sonora se conseguirá mediante la ejecución de un sonido de sirena con licencia CC Attribution 3.0 [37] en los altavoces del monitor que se encontrará conectado a rpi2. Por su parte, para la alerta visual se hará parpadear en rojo la bombilla Wi-Fi mencionada anteriormente. Las alertas visuales en la bombilla tendrán **prioridad** sobre cualquier estado de monitorización de la iluminación en el CPD.

Arquitectura de componentes de rpi3

Por último, se puede apreciar la arquitectura de componentes de rpi3 en la figura 12.

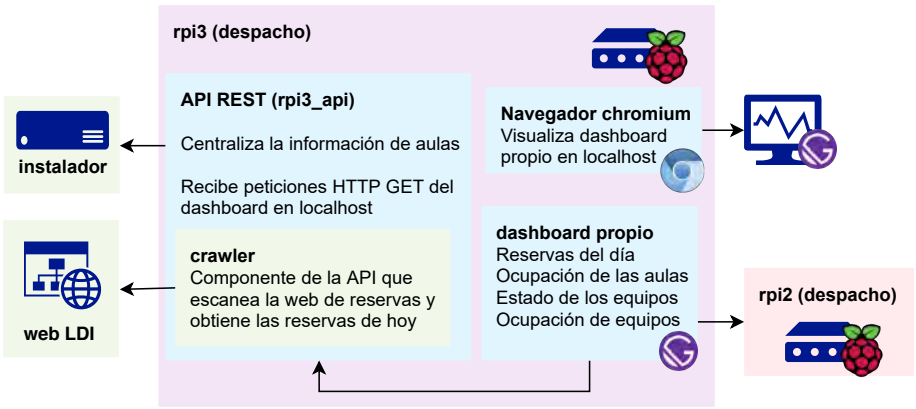

Figura 12: Arquitectura de componentes de la Raspberry Pi 3 (rpi3). Elaboración propia.

Rpi3 ejecutará por una parte un navegador chromium, el cual se cargará al inicio y conectará con un servidor en `localhost` que estará sirviendo el **panel** que se desarrollará con **GatsbyJS**. Este panel servirá de interfaz de usuario para el subsistema de aulas. La proyección se realizará en el monitor conectado a rpi3, que se situará en la pared del despacho y entre las dos puertas que dan acceso al mismo. El servidor será un **servidor ligero de desarrollo propio**, codificado en Go y compilado para ARM.

El dashboard propio, que será codificado con javascript y el framework GatsbyJS, preguntará a rpi2_api por los datos de monitorización del CPD para mostrarlos en el panel inferior de su interfaz, definida más adelante. El resto de datos, razón de ser del **subsistema de aulas**, serán solicitados a rpi3_api (la API REST de rpi3 que estará corriendo en `localhost`). Las solicitudes a las API se realizarán mediante peticiones HTTP GET.

Rpi3_api servirá como **punto central de monitorización de las aulas**. Actualizará la información de estado de los equipos y ocupación (inicios de sesión físicos) de los mismos mediante la ejecución por SSH en el servidor instalador de los **scripts de Lázaro** [11]. Para actualizar los datos de reservas de aulas, rpi3_api parseará la tabla situada en el apartado de **reservas de la web** del LDI con ayuda de un **crawler** que se implementará en la propia API.

Las actualizaciones de los datos se realizarán **cada 60 segundos**, que será la frecuencia con la que el dashboard GatsbyJS realizará las peticiones a rpi3_api. Los datos se encontrarán en la memoria principal de rpi3_api al igual que ocurre con rpi2_api, por contar estos con una alta volatilidad. Estos datos serán proporcionados desde la API hacia el dashboard GatsbyJS en ficheros con formato **JSON** como respuesta a las peticiones HTTP GET. Asimismo, estos datos estarán disponibles para su consulta externa mediante peticiones HTTP GET sin autenticar a rpi3_api.

Diseño de los scripts de monitorización

Los scripts de monitorización de rpi1 se organizarán siguiendo la jerarquía de la figura 13.

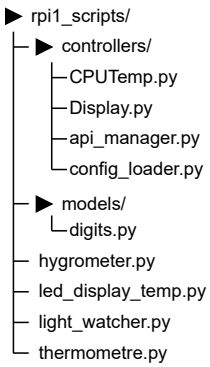

Figura 13: Jerarquía de los scripts de la Raspberry Pi 1 (rpi1). Elaboración propia.

A continuación, se explica brevemente el propósito de cada uno:

- **controllers/:** carpeta que contendrá aquellos scripts genéricos que sirvan para controlar o coordinar aspectos comunes de funcionamiento.
  - **CPUTemp.py:** objeto controlador de la temperatura de la CPU. Proporcionará los métodos necesarios para mitigar o eliminar los errores de medición de temperatura ambiente de los sensores de SenseHat, que pueden verse afectados por el calor que emana la CPU del minicomputador.
  - **Display.py:** objeto controlador del panel led del módulo SenseHat.
  - **api_manager.py:** controlador de las comunicaciones con rpi2_api.
  - **config_loader.py:** controlador de la carga de valores de configuración a partir de ficheros JSON.
- **models/:** contendrá las estructuras y modelos de datos definidos en la implementación.
  - **digits.py:** representación matricial de los dígitos (0-9) para su presentación en el panel led del módulo SenseHat.
- **hygrometer.py:** se encargará de medir la humedad ambiente cada 60 segundos, escribirla a `/tmp/last_hum.txt` y comunicársela a rpi2_api.
- **led_display_temp.py:** se encargará de mantener actualizada la temperatura mostrada en el panel led del módulo SenseHat conforme a las medidas tomadas por `thermometer.py`.
- **light_watcher.py:** se encargará tomar una fotografía para comprobar la iluminación cada 10 segundos y comunicársela a rpi2_api.
- **thermometer.py:** se encargará de medir la temperatura ambiente cada 60 segundos, escribirla a `/tmp/last_temp.txt` y comunicársela a rpi2_api.

Diseños de las API REST

Las API REST a desarrollar seguirán un modelo de jerarquía y arquitectura similar. A continuación se analizan estas, los endpoints de las API, y la configuración de las peticiones y respuestas.

rpi2_api

Los ficheros de rpi2_api se organizarán siguiendo jerarquía de la figura 14.

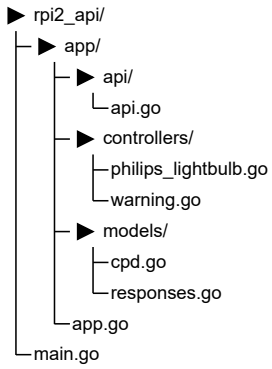

Figura 14: Jerarquía de los ficheros de la Raspberry Pi 2 (rpi2). Elaboración propia.

A continuación, se explica brevemente el propósito de cada uno:

- **app/:** contendrá la implementación de las funcionalidades de rpi2_api.
  - **api/:** contendrá la implementación y estructura básica y necesaria de la API REST.
    - **api.go:** definirá la estructura básica de la API, estableciendo los métodos HTTP y endpoints.
  - **controllers/:** contendrá los controladores de las diferentes funcionalidades del sistema de los que se encargue la API.
    - **philips_lightbulb.go:** proporcionará las funciones y métodos necesarios para manejar la bombilla Wi-Fi Philips® Hue.
    - **warning.go:** proporcionará las funciones y métodos necesarios para gestionar las distintas situaciones de alerta en el CPD y coordinar las alarmas en el despacho.
  - **models/:** contendrá las estructuras y modelos de datos definidos en la implementación.
    - **cpd.go:** objeto CPD que representa el estado en el CPD.
    - **responses.go:** modelos de respuestas a peticiones.
  - **app.go:** inicializará la API, dirección IP y puerto, y levantará los endpoints.
- **main.go:** contendrá el método `main` que inicializará el objeto API y levantará el servidor.

Rpi2_api tendrá definidos los siguientes **endpoints** (en el fichero `api.go`):

- **/:** index de la API REST. Solo aceptará peticiones GET, en cuyo caso retornará un código HTTP 200 y un mensaje en forma de JSON con el siguiente formato:

```
{
    "message": "API is up and running"
}
```

Tabla 72. Respuesta petición GET / rpi2_api.

- **/cpd-status:** información sobre el estado del CPD. Solo aceptará peticiones POST y GET. En el caso de las peticiones GET retornará un código HTTP 200 y un mensaje en forma de JSON con el siguiente formato:

```
{
    "temperature": float,
    "humidity": float,
    "ups status (LDI rack)": "online|battery"
}
```

Tabla 73. Respuesta petición GET /cpd-status rpi2_api.

La petición se podrá **filtrar mediante parámetros en la URL** para que el JSON solo contenga ciertos valores. Estos parámetros son: `/cpd-status?temp&hum&ups`.
Las peticiones POST deberán ser **autenticadas**. La autenticación se hará por medio de un *bearer token* tal que la petición deberá incluir en su cabecera el header `Authorization: Bearer <secret-string>`. El JSON del cuerpo de la petición deberá contener **necesariamente** al menos uno de los siguientes elementos, que se corresponderán con los valores a actualizar.

```
{
    "Temp": float,
    "Hum": float,
    "Light": true|false,
    "UPSStatus": "online|battery"
}
```

Tabla 74. Formato petición POST /cpd-status rpi2_api.

Tras completarse la petición POST, la API responderá con un código HTTP 200 y un mensaje en forma de JSON con el siguiente formato:

```
{
    "message": "OK"
}
```

Tabla 75. Respuesta petición GET /cpd-status rpi2_api.

Por supuesto, los endpoints de la API contarán con **control de errores e información de recuperación** del error. En caso de error nos podremos encontrar con las siguientes situaciones a la hora de intentar obtener una respuesta de la api:

- **Método no permitido:** si se intentase realizar una petición con un método incorrecto (p.e. petición PUT a /cpd-status) la API responderá con un código HTTP 405 y un mensaje en forma de JSON con el siguiente formato:

```
{
    "error": "Method Not Allowed"
}
```

Tabla 76. Respuesta error método no permitido rpi2_api.

- **Error de autenticación 1:** si la petición requiere una autenticación mediante *Bearer token* y dicho header no se encuentra en la cabecera de la petición o el contenido del mismo es una cadena vacía, la API responderá con un código HTTP 401 y un mensaje en forma de JSON con el siguiente formato:

```
{
    "error": "Authorization header not provided or empty"
}
```

Tabla 77. Respuesta error de autenticación 1 rpi2_api.

- **Error de autenticación 2:** si la petición requiere una autenticación mediante *Bearer token* y el token proporcionado es erróneo, la API responderá con un código HTTP 401 y un mensaje en forma de JSON con el siguiente formato:

```
{
    "error": "Unauthorized"
}
```

Tabla 78. Respuesta error de autenticación rpi2_api.

- **Mal formato del cuerpo de la petición:** si la API en una petición POST recibiera un JSON malformado (p.e. sin cerrar alguna llave, falta una coma, etc.) la API responderá con un código HTTP 400 y un mensaje en forma de JSON con el siguiente formato:

```
{
    "error": "Bad Request"
}
```

Tabla 79. Respuesta error mal formato del cuerpo de la petición rpi2_api.

- **Errores internos:** ante cualquier otro error, la API responderá con un código HTTP 500 y un mensaje en forma de JSON con el siguiente formato:

```
{
    "error": "Internal Server Error"
}
```

Tabla 80. Respuesta errores internos rpi2_api.

En este caso, será necesario consultar la **traza de log** de la API REST para conocer mejor la naturaleza del error, que será impresa a **salida estándar**.

rpi3_api

Los ficheros de rpi3_api se organizarán siguiendo jerarquía de la figura 15.

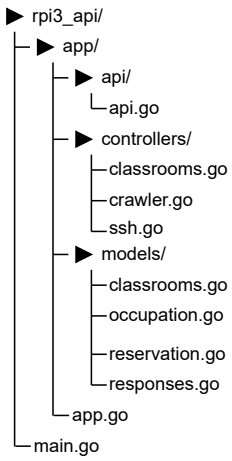

Figura 15: Jerarquía de los ficheros de la Raspberry Pi 3 (rpi3). Elaboración propia.

A continuación, se explica brevemente el propósito de cada uno:

- **app/:** contendrá la implementación de las funcionalidades de rpi3_api.
  - **api/:** contendrá la implementación y estructura básica y necesaria de la API REST.
    - **api.go:** definirá la estructura básica de la API, estableciendo los métodos HTTP y endpoints.
  - **controllers/:** contendrá los controladores de las diferentes funcionalidades del sistema de los que se encargue la API.
    - **classrooms.go:** funciones y métodos necesarios para obtener los datos de estado y ocupación de las aulas
    - **crawler.go:** crawler que obtiene las reservas del día actual desde la web del LDI.
    - **ssh.go:** ejecución de los scripts de Lázaro [11] por SSH en el servidor *instalador*.
  - **models/:** contendrá las estructuras y modelos de datos definidos en la implementación.
    - **classrooms.go:** asignación clave:valor que identifica el estado de cada aula.
    - **occupation.go:** estado de ocupación de los equipos en un aula.
    - **reservation.go:** objeto reserva con los datos de la misma.
    - **responses.go:** modelos de respuestas a peticiones.
  - **app.go:** inicializará la API, dirección IP y puerto, y levantará los endpoints.
- **main.go:** contendrá el método `main` que inicializará el objeto API y levantará el servidor.

Rpi3_api tendrá definidos los siguientes **endpoints** (en el fichero `api.go`):

- **/:** index de la API REST. Solo aceptará peticiones GET, en cuyo caso retornará un código HTTP 200 y un mensaje en forma de JSON con el siguiente formato:

```
{
    "message": "API is up and running"
}
```

Tabla 81. Respuesta GET / rpi3_api.

- /**reservations:** información sobre las reservas diarias. Solo aceptará peticiones GET, en cuyo caso retornará un código HTTP 200 y un mensaje en forma de JSON con el siguiente formato:

```
[
    {
        "Classroom": "string",
        "Subject": "string",
        "Professor": "string",
        "Datetime": "string"
    },
    ...
]
```

Tabla 82. Respuesta GET /reservations rpi3_api.

  En la lista del JSON se incluirán todas las **reservas programadas para el día de actual** independientemente de la hora actual. La lista del JSON puede estar vacía si no hay reservas programadas para ese día.

- /**classrooms:** información sobre el estado actual de las aulas. Solo aceptará peticiones GET, en cuyo caso retornará un código HTTP 200 y un mensaje JSON con el siguiente formato:

```
{
    "F16": int,
    "F18": int,
    "C05": int,
    "C06": int
}
```

Tabla 83. Respuesta GET /classrooms rpi3_api.

  Estos valores numéricos podrán ser:

  - **0:** el aula se encuentra **libre** para que cualquier estudiante la utilice o los técnicos realicen mantenimiento.
  - **1:** el aula está **ocupada** por una reserva.
  - **2:** el aula tiene una **reserva** que empezará **en 30 minutos** o menos.
  - **3:** el aula tiene una **reserva** que empezará **en 10 minutos** o menos.

- /**occupation:** información sobre el estado y la ocupación de los equipos de las aulas. Solo aceptará peticiones GET, en cuyo caso retornará un código HTTP 200 y un mensaje en forma de JSON con el siguiente formato:

```
{
    "F16": {
        "Linux": int,
        "Windows": int,
        "Shutdown": int,
        "TimeOut": int,
        "LoginsLinux": int,
        "LoginsWindows": int,
        "Computers": [
            int, int, int, ...
        ]
    }, ...
}
```

Tabla 84. Respuesta GET /occupation rpi3_api.

Para cada aula, los seis primeros atributos indicarán el número de equipos asociados a dicho atributo. La lista `Computers` será una lista ordenada donde cada índice i representará el equipo i del aula. Los valores indicarán el **estado** de dicho equipo. Estos valores podrán ser:

  - **0:** el equipo se encuentra **apagado**.
  - **1:** el equipo está **encendido** y está ejecutando el sistema operativo **GNU/Linux**.
  - **2:** el equipo está **encendido** y está ejecutando el sistema operativo **Windows**.
  - **3:** el equipo está **encendido** con **GNU/Linux** y un **usuario** está usando el equipo.
  - **4:** el equipo está **encendido** con **Windows** y un **usuario** está usando el equipo.
  - **5:** el equipo está en estado de **error** o da *time out* al conectar.

Por supuesto, los endpoints de la API contarán con **control de errores e información de recuperación** del error. En caso de error nos podremos encontrar con las siguientes situaciones a la hora de intentar obtener una respuesta de la api:

- **Método no permitido:** si se intentase realizar una petición con un método incorrecto (p.e. petición POST a /classrooms) la API responderá con un código HTTP 405 y un mensaje en forma de JSON con el siguiente formato:

```
{
    "error": "Method Not Allowed"
}
```

Tabla 85. Respuesta error método no permitido rpi3_api.

- **Errores internos:** ante cualquier otro error, la API responderá con un código HTTP 500 y un mensaje en forma de JSON con el formato de la tabla 86.
  En este caso, será necesario consultar la **traza de log** de la API REST para conocer mejor la naturaleza del error, que será impresa a **salida estándar**.

```
{
    "error": "Internal Server Error"
}
```

Tabla 86. Respuesta errores internos rpi3_api.

Diseños de las interfaces de usuario

Se establece el diseño de la interfaz de los dos subsistemas en base a los **requisitos** de usuario recogidos en el capítulo anterior y los posteriores requisitos software identificados a partir de estos. El subsistema del **CPD** contará con un **dashboard de Grafana**. El subsistema de **aulas** y equipos contará con un **dashboard propio** desarrollado con el framework **GatsbyJS** y react. Se definen así los siguientes prototipos o *mockups*.

Dashboard Grafana para el subsistema del CPD

Como se puede apreciar en la figura 16, el panel contará con tres indicadores en forma de velocímetro o *gauge* y tres gráficas, para cumplir con los requisitos del sistema.

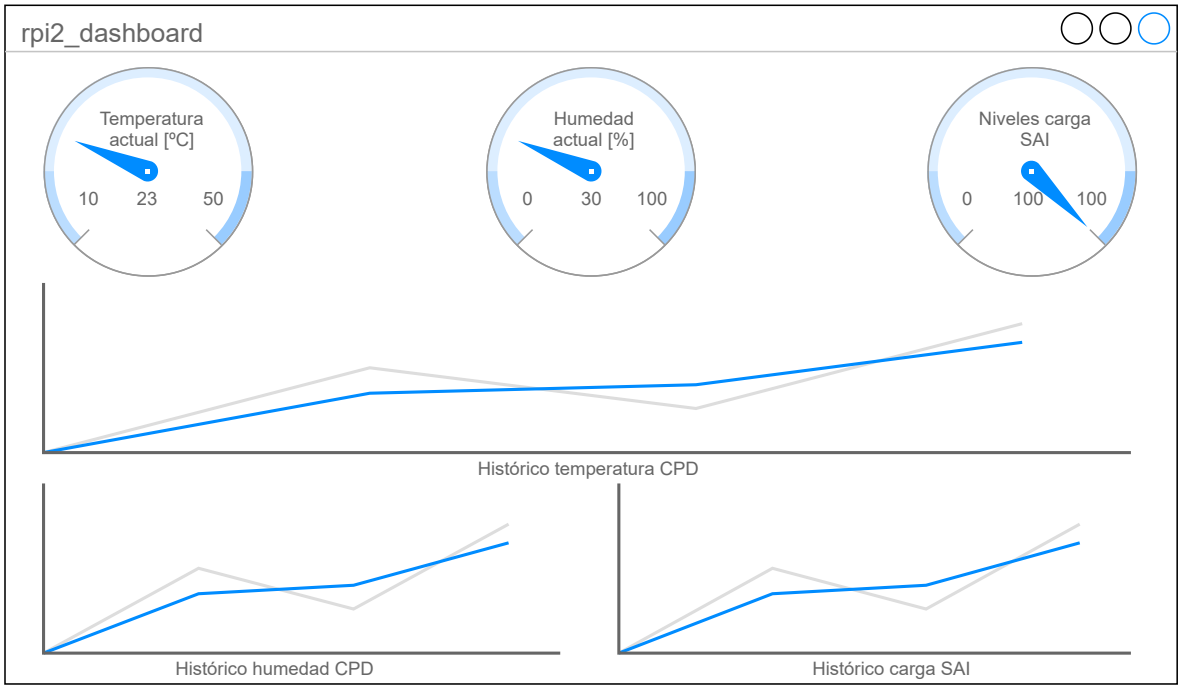

Figura 16: Prototipo de la interfaz de usuario del subsistema del CPD. Panel de Grafana. Elaboración propia.

Los indicadores de velocímetro se situarán en la parte superior, en una única fila. De izquierda a derecha mostrarán los valores **actuales** de temperatura y humedad en el CPD y niveles de carga del SAI en el rack del LDI.

En la parte central se encontrará un gráfico de líneas con el **histórico de temperatura** dentro del CPD en la última semana (7 días naturales). En la parte inferior izquierda se situará asimismo un gráfico de líneas con el **histórico de humedad** dentro del CPD en la última semana.

Por último, en la parte inferior derecha se dispondrá un gráfico de líneas con dos gráficas superpuestas. Por un lado, el **histórico del nivel de carga del SAI** en la última semana; y por otro lado, el **histórico del tiempo de servicio** que nos proporcionaría la batería del SAI para la carga de trabajo en ese momento.

Para la proyección en el monitor que se situará sobre la puerta del CPD se mostrarán únicamente, **de forma individual y a pantalla completa**, los tres velocímetros y el histórico de temperatura. Estos elementos irán rotando automáticamente **cada 10 segundos** como si de una presentación se tratase.

Dashboard GatsbyJS para el subsistema de aulas

El subsistema de aulas y equipos tiene **dos vistas principales**: la vista de **reservas** definida en el prototipo de la figura 17, y la vista de **aulas** definida en el prototipo de la figura 18.

La interfaz contará con un **aspecto consistente entre las diversas vistas**. Esto ayudará al usuario a en todo momento comprender dónde debe buscar qué elemento. El diseño será coherente con lo establecido en los requisitos del sistema. Así pues, contará con los elementos que se pueden apreciar en cualquiera de los dos *mockups* (ilustraciones 17 y 18). Estos serán:

- Un **panel superior**, con el nombre del servicio y la fecha y hora actual.
- Un **panel inferior**, con información del CPD resumida que se traslada de derecha a izquierda.
- Un **panel principal** a la izquierda, que mostrará la información de cada vista.
- Un **panel lateral** a la derecha, con la información del estado y ocupación de las aulas.

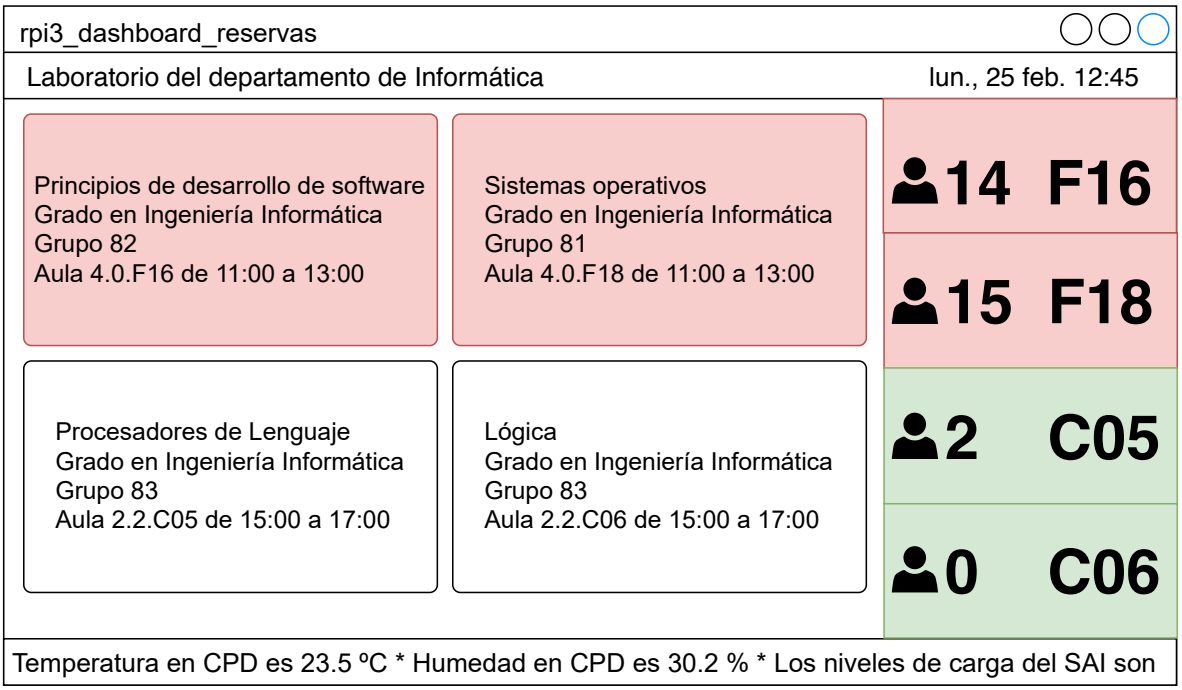

Figura 17: Prototipo de la interfaz de usuario del subsistema de aulas. Vista de reservas. Elaboración propia.

El **panel lateral** mostrará, conforme a los requisitos del sistema, el número de inicios de sesión físicos en el aula y el estado de ocupación o reserva del aula mediante el código de colores definido en los requisitos (RF-16: Visualización del estado de las aulas.).

Para la **vista de reservas** (figura 17) se mostrarán en el panel principal hasta un máximo de cuatro tarjetas con información de las reservas. Si la reserva estuviera en curso la tarjeta se encontrará coloreada en rojo. La información de las tarjetas y demás atributos será conforme a los requisitos del sistema. Si ese día no hubiera reservas o todas hubiesen finalizado, se indicará asimismo en una tarjeta que ocupará todo el panel principal.

El prototipo para la **vista de aulas** se puede apreciar en la figura 18. Durante esta vista, en el panel principal aparecerán tantas tarjetas como equipos disponga el aula. Estos equipos se organizarán siguiendo la disposición de las aulas, conforme a los requisitos del sistema.

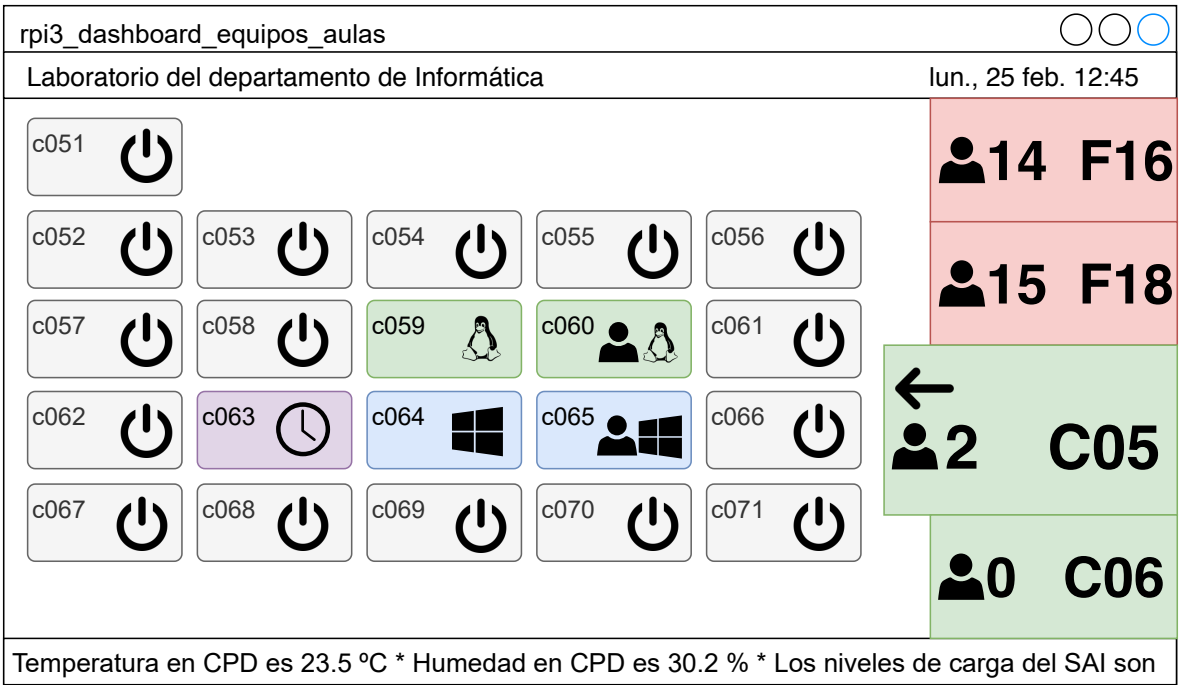

Figura 18: Prototipo de la interfaz de usuario del subsistema de aulas. Vista de aulas. Elaboración propia.

Cada equipo mostrará un **icono y un color identificativo** de su estado, así como un icono de usuario si alguien se encuentra utilizando físicamente el equipo. En la figura 18 se aprecia el diseño de los seis posibles estados definidos en los requisitos del sistema.

Además, el panel lateral se aprovechará para servir como **indicador de qué aula se está visualizando**. Para ello, el aula actual se destacará con un mayor tamaño (especialmente en ancho) y el icono de una flecha apuntando a la vista para señalar su significado.

Por último, si la información de cualquiera de los tres paneles (principal con su correspondiente vista, lateral o inferior) no se pudiera recuperar del sistema, la interfaz sustituirá el panel por un **mensaje de error** resaltado en rojo como el que se muestra en la figura 19. El panel principal contendrá el mensaje correspondiente al error de la vista actual.

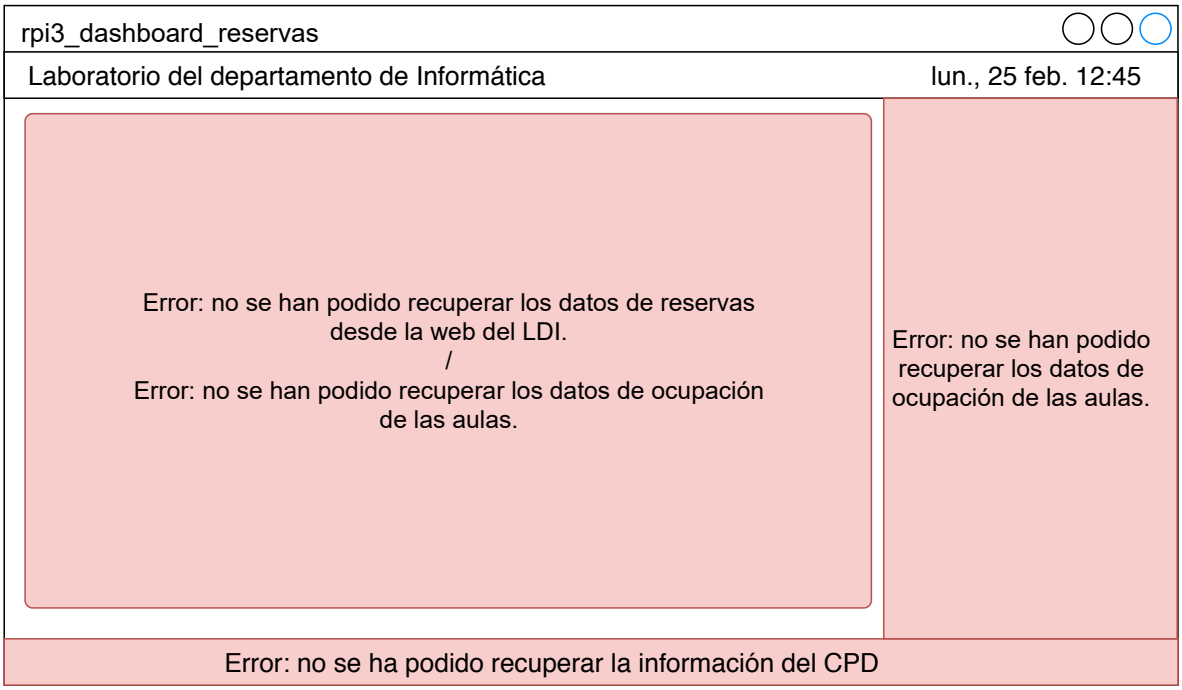

Figura 19: Prototipo de la interfaz de usuario del subsistema de aulas. Vista de error. Elaboración propia.

Implementación

El código fuente y demás elementos software del sistema desarrollado se encuentran en un **repositorio git** en un servidor de la infraestructura del LDI. Estos componentes, eliminando claves y tokens serán liberados y el repositorio duplicado y accesible públicamente desde GitHub [28] una vez sea publicada esta memoria en la biblioteca de la universidad.

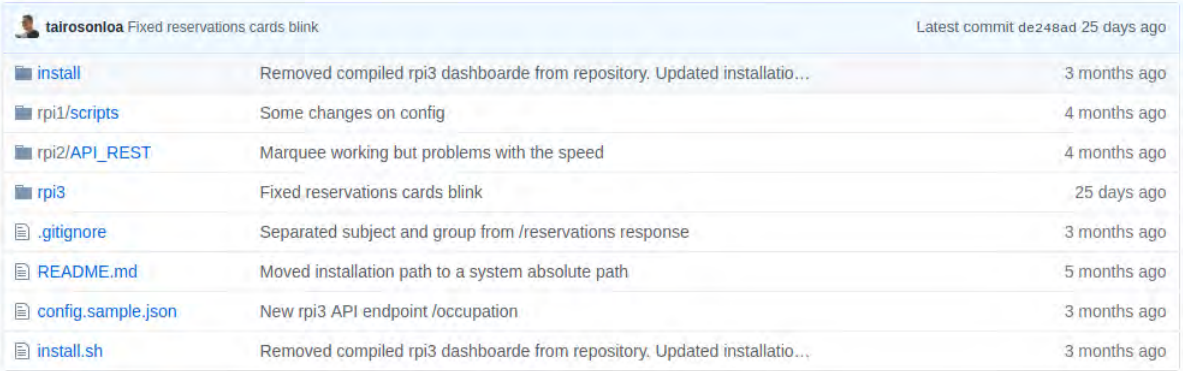

Figura 20: Repositorio git del proyecto en GitHub [28]. Captura de pantalla.

En la figura 20 se puede apreciar la disposición de los elementos en el repositorio, donde cada componente está en una carpeta separada. La carpeta `install/` contiene todos los elementos necesarios por el script de instalación `install.sh`. El contenido de las carpetas `rpi<N>/` obedece a la jerarquía establecida en el capítulo de diseño.

Figura 21: Estadísticas de lenguajes utilizados en el proyecto según GitHub. Captura de pantalla.

Como se puede apreciar en la figura 21, el proyecto tiene carácter multilenguaje, estando constituido en un 34.5 % de código escrito en Go, 24.7 % código JavaScript, 16.8 % de código Python 3 y un 14.7 % de código de shell Bash.

Language	files	blank	comment	code
Go	18	119	143	869
JavaScript	6	35	68	494
Python	9	88	87	335
CSS	4	49	2	305
Bourne Shell	1	42	39	226
make	3	6	18	66
Bourne Again Shell	1	7	3	27
JSON	1	0	0	12
SUM:	43	346	360	2334

Figura 22: Estadísticas de lenguajes utilizados en el proyecto según cloc. Captura de pantalla.

Tras analizar el repositorio con la herramienta cloc [38] como se muestra en la figura 22 se puede apreciar el sistema implementado cuenta con **un total de 2 334 líneas de código**.

Implantación

El proceso de implantación cuenta de **dos fases** principales, la **preparación de la infraestructura** que comprende el despliegue de la misma y la instalación de dependencias, y la **instalación y configuración** de los componentes del sistema.

Preparación de la infraestructura del sistema

El **entorno operacional** del sistema consta de los siguientes elementos:

- Minicomputador Raspberry Pi modelo 3B+ (**rpi1**) dentro del CPD. Conectado por ethernet a la subred 163.117.170.0/24 y con IP 163.117.170.101.
- Minicomputador Raspberry Pi modelo 3B+ (**rpi2**) en el despacho 4.0.F11, sobre la puerta del CPD. Conectado por ethernet a la subred 163.117.170.0/24 y con IP 163.117.170.102.
- Minicomputador Raspberry Pi modelo 3B+ (**rpi3**) en el despacho 4.0.F11, en la pared principal. Conectado por ethernet a la subred 163.117.170.0/24 y con IP 163.117.170.103.
- Monitor de 23 pulgadas conectado por HDMI a rpi2.
- Monitor de 23 pulgadas conectado por HDMI a rpi3.
- Infraestructura heredada (servidores instalador, ultraheroe y web. Aulas docentes).

Antes de proceder a la instalación y configuración de los componentes del sistema, es preciso instalar las **dependencias** necesarias en los distintos minicomputadores Raspberry Pi. Se asume que todos los minicomputadores cuentan con sistema operativo **Raspbian Stretch Lite** [39] (basado en Debian Strech, GNU/Linux). A continuación, se listan las dependencias necesarias así como la versión mínima de las mismas que aseguran el correcto funcionamiento del sistema. Estas versiones se encuentran en los repositorios estables a la hora de documentar el proceso de implantación.

Dependencias necesarias en rpi1

- `iptables-persistent=1.0.4` (para mantener las reglas del cortafuegos tras reiniciar).
- `python=3.5.3` (para ejecutar los scripts).
- `python3-numpy=1.12.1` (para trabajar eficientemente con los píxeles de la cámara).
- `python3-picamera=1.13` (para controlar la cámara desde los scripts).
- `python3-sense-hat=2.2.0` (para controlar el módulo SenseHat desde los scripts).
- `zabbix-agent=3.0.7` (para comunicarse con la instancia de Zabbix en ultraheroe).

Dependencias necesarias en rpi2

- `iptables-persistent=1.0.4` (para mantener las reglas del cortafuegos tras reiniciar).
- `chromium-browser=72.0.3626.121` (para mostrar el dashboard de Grafana).
- `lightdm=1.18.3` (para iniciar sesión gráfica automáticamente al encender rpi2).
- `openbox=3.6.1` (entorno de escritorio minimalista. Permite lanzar chromium al inicio).
- `omxplayer=2018.09.05` (para reproducir sonidos de alarma desde la API).

Dependencias necesarias en rpi3

- `iptables-persistent=1.0.4` (para mantener las reglas del cortafuegos tras reiniciar).
- `chromium-browser=72.0.3626.121` (para mostrar el dashboard de gatsby).
- `lightdm=1.18.3` (para iniciar sesión gráfica automáticamente al encender rpi3).
- `openbox=3.6.1` (entorno de escritorio minimalista. Permite lanzar chromium al inicio).
- `nodejs=8.11.1` (para compilar el dashboard de gatsby).
- `npm=5.8.0` (para instalar las dependencias durante la compilación del dashboard de gatsby).

Instalación y configuración de los componentes del sistema

Cumpliendo con los requisitos del sistema se ha desarrollado un script de instalación `install.sh`. Este script debe ejecutarse como **root** en cada uno de los minicomputadores Raspberry Pi. Durante su ejecución **preguntará al usuario** qué componente del sistema está tratando de instalar (rpi1, rpi2, rpi3). Asimismo, **generará la configuración** necesaria para el correcto funcionamiento del sistema, mostrando al usuario los valores por defecto de la misma y otorgándole la posibilidad de modificarlos durante el proceso de instalación.

El sistema quedará instalado y será ejecutado bajo el usuario lab, un usuario sin privilegios por **mayor seguridad**. Los permisos de root son necesarios para mover los binarios y demás componentes del sistema a la ruta correspondiente dentro de la jerarquía de ficheros de los sistemas *UNIX like*, así como instalar las dependencias necesarias y configurar las reglas del cortafuegos, entre otras tareas.

A pesar de que el mencionado script `install.sh` cubre todo el proceso de instalación y configuración del sistema de una forma interactiva y automatizada, este proceso se detalla en los apartados subsiguientes.

Proceso de instalación

El proceso de instalación del sistema se compone de algunas fases comunes que se detallan a continuación:

1. **Instalación de las dependencias** descritas en el apartado anterior para cada componente.
2. **Configuración del cortafuegos.** Establecimiento de las reglas de seguridad del cortafuegos.
3. **Instalación del componente.** Colocación de scripts y binarios en el directorio de instalación.
4. **Activación de componentes y permisos.** Activación de funcionalidades y componentes hardware para su uso por parte del usuario de instalación lab, que ejecutará el sistema.
5. **Configuración del arranque automático** de los componentes del sistema.
6. **Configuración de los componentes del sistema.** Dependiente del componente.

Seguidamente se especifican las peculiaridades del proceso de instalación del sistema para cada minicomputador Raspberry Pi.

Proceso de instalación de rpi1

En primer lugar se procede a la instalación de las **dependencias** necesarias, comprobando que estas cumplen con los requisitos mínimos de versión especificados en el apartado Dependencias necesarias en rpi1. Estas versiones se encuentran en los repositorios estables a la hora de documentar el proceso de implantación.

```
#> apt-get update && apt-get upgrade
#> apt-get install iptables-persistent python-numpy python3-picamera python3-
    sense-hat zabbix-agent
```

Una vez instaladas las dependencias, se procede a la configuración del **cortafuegos**.

Se definen las políticas por defecto. Se acepta todo el tráfico saliente y se desecha todo el tráfico entrante o de paso que no esté definido explícitamente en las reglas del cortafuegos.

```
#> iptables -F
#> iptables -P INPUT DROP
#> iptables -P OUTPUT ACCEPT
#> iptables -P FORWARD DROP
```

Se permiten las conexiones activas, la interfaz *loopback*, la respuesta a llamadas *ping* y el acceso desde las subredes del LDI al puerto 22 (SSH).

```
#> iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
#> iptables -I INPUT 1 -i lo -j ACCEPT
#> iptables -A INPUT -p icmp --icmp-type 8 -j ACCEPT
#> iptables -A INPUT -p tcp --dport 22 -s 163.117.170.0/24 -j ACCEPT
#> iptables -A INPUT -p tcp --dport 22 -s 163.117.142.0/24 -j ACCEPT
```

Asimismo, se definen las reglas propias de este componente y se asegura su persistencia tras el reinicio del sistema. Se permite el acceso de **ultraheroe** para comunicarse con el agente de Zabbix.

```
#> iptables -A INPUT -p tcp --dport 10050 -s 163.117.142.235/32 -j ACCEPT
#> iptables-save > /etc/iptables/rules.v4
```

A continuación se procede con la instalación del componente. Se instalan los **scripts** de rpi1 en el directorio `/usr/local/bin/rpi1_cpd/`. Se proporciona la autoría de dicha carpeta y subcarpetas al usuario de instalación lab.

Después de instalar los scripts se activan los **componentes hardware** necesarios. Rpi1 requiere de acceso a la interfaz de la cámara y a la interfaz i2c para comunicarse con el módulo SenseHat.

```
#> raspi-config nonint do_i2c 0
#> raspi-config nonint do_camera 0
```

Asimismo, se requiere que el usuario de instalación pertenezca al grupo *video* para poder utilizar la cámara, y a los grupos *i2c* e *input* para utilizar el módulo SenseHat. Esto es debido al anillo de **permisos** en el que se basa la seguridad de los sistemas de ficheros UNIX like.

```
#> usermod -aG input lab
#> usermod -aG i2c lab
#> usermod -aG video lab
```

Una vez que el componente está instalado y se han activado los elementos y otorgado los permisos necesarios para su funcionamiento, se procede a configurar su **arranque automático**. Para ello, se copian los siguientes ficheros a la ruta `/etc/systemd/system/` desde la carpeta `install/`:

- `cdp_temperature.service`
- `cdp_humidity.service`
- `cdp_light.service`

Estos demonios aseguran que los scripts se iniciarán en el arranque del sistema operativo, así como que estos se relanzarán en caso de que abortaran su ejecución por algún error. Para activar los demonios se procede con el cliente de terminal de systemd.

```
#> systemctl enable cdp_temperature.service
#> systemctl enable cdp_humidity.service
#> systemctl enable cdp_light.service
#> systemctl enable zabbix-agentd.service
```

Para finalizar, rpi1 requiere de la configuración del **agente de Zabbix**. Se modifican o se añaden los siguientes atributos en el fichero `/etc/zabbix/zabbix_agentd.conf`:

```
Server=163.117.142.235
ServerActive=163.117.142.235
UserParameter=cpd.temp, /bin/cat /tmp/last_temp.txt
UserParameter=cpd.hum, /bin/cat /tmp/last_hum.txt
```

Llegados a este punto, es necesario generar el **fichero de configuración** del sistema antes de continuar. Todo lo relativo acerca de las sintaxis y valores de este fichero queda explicado más adelante en el apartado Configuración de los componentes.

Con los pasos anteriores queda instalado el componente rpi1. Para ponerlo en funcionamiento basta con reiniciar el equipo o reiniciar manualmente los demonios que se han activado anteriormente. Sin embargo, **se requiere la instalación de rpi2 para poner en funcionamiento rpi1**.

Proceso de instalación de rpi2

En primer lugar se procede a la instalación de las **dependencias** necesarias, comprobando que estas cumplen con los requisitos mínimos de versión especificados en el apartado Dependencias necesarias en rpi2. Estas versiones se encuentran en los repositorios estables a la hora de documentar el proceso de instalación.

```
#> apt-get update && apt-get upgrade
#> apt-get install iptables-persistent openbox chromium-browser lightdm omxplayer
```

Una vez instaladas las dependencias, se procede a la configuración del **cortafuegos**. Como con rpi1, se definen las mismas políticas por defecto y se permiten las conexiones activas, la interfaz *loopback*, la respuesta a llamadas *ping* y el acceso desde las subredes del LDI al puerto 22 (SSH).

Asimismo, se definen las reglas propias de este componente y se asegura su persistencia tras el reinicio del sistema. Se abre el puerto destinado a la escucha de **rpi2_api** para todas las subredes de la Universidad Carlos III de Madrid, por si el personal del Departamento de Informática quisiera comprobar el estado del CPD en cualquier momento desde sus respectivos despachos.

```
#> iptables -A INPUT -p tcp --dport 3000 -s 163.117.0.0/16 -j ACCEPT
#> iptables-save > /etc/iptables/rules.v4
```

A continuación se procede con la **instalación** del componente. Se instalan el **binario** de rpi2_api en el directorio `/usr/local/bin/` y se le proporciona los permisos en octal `755`. Asimismo se instala el fichero MP3 con el sonido que de alarma en el directorio `/usr/local/share/`.

Después se activan los **componentes hardware** necesarios. Rpi2 requiere de acceso a la interfaz HDMI para hacer uso del monitor externo. Para ello, es preciso que el usuario de instalación pertenezca al grupo *video*.

```
#> usermod -aG video lab
```

Una vez que el componente está instalado y se han activado los elementos y otorgado los permisos necesarios para su funcionamiento, se procede a configurar su **arranque automático**. Para ello, se copia el fichero `rpi2_api.service` desde `install/` a la ruta `/etc/systemd/system/` y se procede a su activación desde systemd.

```
#> systemctl enable rpi2_api.service
```

Asimismo hay que configurar el **inicio de sesión gráfica automático** y la ejecución del navegador chromium con el dashboard de Grafana. En primer lugar, se activa el inicio de sesión y se desactiva el visionado del cursor en la pantalla.

```
#> raspi-config nonint do_boot_behaviour B4
#> sed -i "s/#xserver-command=X/xserver-command=X -nocursor/g" /etc/lightdm/
    lightdm.conf
```

Para la ejecución del navegador **chromium a pantalla completa al inicio**, con el dashboard de Grafana en modo kiosko, se modifica el fichero `/etc/xdg/openbox/autostart`, el cual debe tener permisos de ejecución. Este fichero debe contener las siguientes líneas:

```
sed -i 's/"exited_cleanly":false/"exited_cleanly":true/' ~/.config/chromium/'
    Local State'
sed -i 's/"exited_cleanly":false/"exited_cleanly":true/; s/"exit_type":"[^"]\+"/"
    exit_type":"Normal"/' ~/.config/chromium/Default/Preferences
/usr/bin/chromium-browser --disable-infobars --kiosk 'http://ultraheroe.lab.inf.
    uc3m.es:3000/playlists/play/4?kiosk'
```

Para terminar de configurar el **monitor**, se define su resolución.

```
#> sed -i "s/#disable_overscan=1/disable_overscan=1/g" /boot/config.txt
#> sed -i "s/#overscan_left=16/overscan_left=0/g" /boot/config.txt
#> sed -i "s/#overscan_right=16/overscan_right=0/g" /boot/config.txt
#> sed -i "s/#overscan_top=16/overscan_top=0/g" /boot/config.txt
#> sed -i "s/#overscan_bottom=16/overscan_bottom=0/g" /boot/config.txt
#> sed -i "s/#framebuffer_width=1280/framebuffer_width=1920/g" /boot/config.txt
#> sed -i "s/#framebuffer_height=720/framebuffer_height=1080/g" /boot/config.txt
```

Y se configura el **encendido y apagado automático** del mismo mediante tareas *cron job*. Estas tareas ejecutarán el script `raspi-monitor`, un sencillo script que activa o desactiva la señal enviada a la interfaz HDMI. Este estará alojado en `/usr/local/sbin/` y deberá tener permisos de ejecución.

```
#> (crontab -l 2>/dev/null; echo "10 8 * * 1-5 /usr/local/sbin/raspi-monitor on >
    /dev/null 2>&1") | crontab -
#> (crontab -l 2>/dev/null; echo "10 21 * * 1-5 /usr/local/sbin/raspi-monitor off
    > /dev/null 2>&1") | crontab -
```

Llegados a este punto, es necesario generar el **fichero de configuración** del sistema antes de continuar. Todo lo relativo acerca de las sintaxis y valores de este fichero queda explicado más adelante en el apartado Configuración de los componentes.

Para finalizar, rpi2 requiere ser emparejada por primera vez con la unidad **Philips® Hue** bridge para controlar la bombilla Wi-Fi Philips®. Para ello, se debe pulsar el botón central del bridge y a continuación iniciar rpi2_api dentro de los siguientes 30 segundos.

```
#> systemctl start rpi2_api.service
```

Tras esto, el componente rpi2 estará funcionando. Gracias al demonio configurado, rpi2_api arrancará automáticamente al inicio del sistema y si su ejecución se abortara por cualquier motivo. Ahora que rpi2 se encuentra en funcionamiento se pueden iniciar los componentes de rpi1 como se indican al final del apartado anterior.

Proceso de instalación de rpi3

En primer lugar se procede a la instalación de las **dependencias** necesarias, comprobando que estas cumplen con los requisitos mínimos de versión especificados en el apartado Dependencias necesarias en rpi3. Estas versiones se encuentran en los repositorios estables a la hora de documentar el proceso de instalación.

```
#> apt-get update && apt-get upgrade
#> apt-get install iptables-persistent openbox chromium-browser lightdm nodejs
    npm
```

Una vez instaladas las dependencias, se procede a la configuración del **cortafuegos**. Al igual que con rpi1 y rpi2, se definen las mismas políticas por defecto y se permiten las conexiones activas, la interfaz *loopback*, la respuesta a llamadas *ping* y el acceso desde las subredes del LDI al puerto 22 (SSH).

Asimismo, se definen las reglas propias de este componente y se asegura su persistencia tras el reinicio del sistema. Se abre el puerto destinado a la escucha de **rpi3_api** para todas las subredes de la Universidad Carlos III de Madrid, por si el personal del Departamento de Informática quisiera comprobar el estado de las aulas en cualquier momento desde sus respectivos despachos. Asimismo, se abre la **interfaz gráfica** del sistema a la subred del laboratorio en la que se encuentran los equipos de los técnicos.

```
#> iptables -A INPUT -p tcp --dport 3000 -s 163.117.0.0/16 -j ACCEPT
#> iptables -A INPUT -p tcp --dport 9000 -s 163.117.142.0/24 -j ACCEPT
#> iptables-save > /etc/iptables/rules.v4
```

A continuación se procede con la **instalación** del componente. Se instalan los **binarios** de rpi3_api y el servidor desarrollado en el directorio `/usr/local/bin/` y se le proporcionan los permisos en octal `755`. Se copian los estáticos del dashboard gatsby en la ruta `/srv/rpi3/` y se modifica la autoría de dicha carpeta y subcarpetas al usuario de instalación lab.

Si se quisiera volver a generar los **estáticos del dashboard** de gatsby será preciso ejecutar los siguientes comando desde la carpeta que contiene el proyecto de react.

```
#> npm install
#> npm run build
```

Se generará una carpeta `public/` que contendrá los estáticos del dashboard.

Después se activan los **componentes hardware** necesarios. Rpi3 requiere de acceso a la interfaz HDMI para hacer uso del monitor externo. Para ello, es preciso que el usuario de instalación pertenezca al grupo *video*.

```
#> usermod -aG video lab
```

Una vez que los componentes están instalados y se han activado los elementos y otorgado los permisos necesarios para su funcionamiento, se procede a configurar su **arranque automático**. Para ello, se copian los ficheros `rpi3_api.service` y `rpi3_web_server.service` desde `install/` a la ruta `/etc/systemd/system/` y se procede a su activación desde systemd.

```
#> systemctl enable rpi3_api.service
#> systemctl enable rpi3_web_server.service
```

Asimismo hay que configurar el **inicio de sesión gráfica automático** y la ejecución del navegador chromium con el dashboard de gatsby. En primer lugar, se activa el inicio de sesión y se desactiva el visionado del cursor en la pantalla de la misma forma que para rpi2.

```
#> raspi-config nonint do_boot_behaviour B4
#> sed -i "s/#xserver-command=X/xserver-command=X -nocursor/g" /etc/lightdm/
   lightdm.conf
```

Para la ejecución del navegador **chromium a pantalla completa al inicio**, que muestre el dashboard desarrollado con GatsbyJS, se modifica el fichero `/etc/xdg/openbox/autostart`, el cual debe tener permisos de ejecución. Este fichero debe contener las siguientes líneas:

```
sed -i 's/"exited_cleanly":false/"exited_cleanly":true/' ~/.config/chromium/'
    Local State'
sed -i 's/"exited_cleanly":false/"exited_cleanly":true/; s/"exit_type":"[^"]\+"/"
    exit_type":"Normal"/' ~/.config/chromium/Default/Preferences
/usr/bin/chromium-browser --disable-infobars --kiosk 'http://localhost:9000'
```

Para terminar de configurar el **monitor** se define su resolución y se configura el **encendido y apagado automático** del mismo al igual que se hizo para rpi2.

Por último se debe generar un par de **claves RSA** en rpi3, necesarias para conectarse por SSH al servidor instalador y ejecutar los scripts de Lázaro [11] para comprobar la ocupación de las aulas. Para generar la clave se ejecuta el siguiente comando.

```
su lab -c "ssh-keygen -t rsa -b 4096"
```

Se generarán dos ficheros, `/home/lab/.ssh/id_rsa` y `/home/lab/.ssh/id_rsa.pub`. El contenido de del segundo (**clave pública**) deberá ser volcado al fichero `/root/.ssh/authorized_keys` de instalador.

Llegados a este punto, es necesario generar el **fichero de configuración** del sistema antes de continuar. Todo lo relativo acerca de las sintaxis y valores de este fichero queda explicado más adelante en el apartado Configuración de los componentes.

Con los pasos anteriores el componente queda instalado y preparado para su uso. Para activarlo, basta con reiniciar el minicomputador o lanzar los demonios manualmente.

```
#> systemctl start rpi3_api.service
#> systemctl start rpi3_web_server.service
```

Configuración de los componentes

Los distintos componentes y subsistemas obtiene sus valores de configuración desde un **fichero JSON** que es instalado en la ruta `/etc/rpi{1,2,3}_conf.json`. El número en el nombre del fichero vendrá determinado por el componente al cual pertence la configuración.

Si se utiliza el **script de instalación** para desplegar el sistema, este script genera el fichero, asigna los valores indicados por el usuario o los valores por defecto en caso contrario, e instala la configuración en la ruta adecuada.

A continuación se muestra la sintaxis del fichero de configuración JSON y los valores por defecto, a excepción de los tokens y demás información sensible.

```
{
    "Rpi2APIAddress"          : "163.117.170.102",
    "Rpi2APIPort"             : 3000,
    "Rpi3APIAddress"          : "163.117.170.103",
    "Rpi3APIPort"             : 3000,
    "Rpi2APIAuthorizedToken" : "Bearer XXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "HueBridgeAddress"        : "XXX.XXX.XXX.XXX",
    "HueBridgeToken"          : "XXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "AlarmSoundPath"          : "/usr/local/share/alarm.mp3",
    "ControlServer"           : "instalador.lab.inf.uc3m.es:22",
    "OccupationCmd"           : "comprobar_ocupacion.py --au"
}
```

Tabla 87. Sintaxis y valores por defecto fichero de configuración JSON.

Seguidamente se explica el significado de cada valor y qué componentes los requieren en su fichero JSON de configuración. Cualquier valor extra o innecesario simplemente será ignorado por el componente.

- **Rpi2APIAddress:** dirección IP en la que escucha rpi2_api. Requerido por rpi1, rpi2 y rpi3.
- **Rpi2APIPort:** puerto en el que escucha rpi2_api. Requerido por rpi1, rpi2 y rpi3.
- **Rpi3APIAddress:** dirección IP en la que escucha rpi3_api. Requerido únicamente por rpi3.
- **Rpi3APIPort:** puerto en el que escucha rpi3_api. Requerido únicamente por rpi3.
- **Rpi2APIAuthorizedToken:** token bearer de autenticación para las peticiones POST a las API. Requerido por rpi1 (remitente) y rpi2 (destinatario).
- **HueBridgeAddress:** dirección IP del Philips® Hue bridge. Requerido por rpi2.
- **HueBridgeToken:** token de emparejamiento con el Philips® Hue bridge. Requerido por rpi2.
- **AlarmSoundPath:** directorio de instalación del sonido de la alarma. Requerido por rpi2.
- **ControlServer:** dirección (IP o nombre) y puerto SSH del servidor de control *instalador*. Requerido por rpi3.
- **OccupationCmd:** comando a ejecutar en instalador para conocer la ocupación de las aulas (scripts de Lázaro [11]). Requerido por rpi3.

Carga de datos

Dado que el sistema no tiene almacenamiento persistente, y dado que los datos que este maneja son volátiles (se actualizan cada 60 segundos), **no se aplica carga inicial de datos** al sistema.

Prueba de despliegue completa

Una vez finalizada la fase de desarrollo del sistema, se procede con la **prueba de despliegue completa** definida en la planificación del proyecto. Se formatean los tres minicomputadores Raspberry Pi, se instala el sistema operativo Raspbian Stretch Lite, se crea el usuario sin privilegios `lab` y se conectan las raspberrys a sus respectivos componentes hardware y direcciones IP.

Tras estos preparativos, se ejecuta el script de instalación `install.sh` en cada uno de los mini-computadores y tras su ejecución se comprueba si se cumplen las siguientes condiciones:

Tarea	Estado
¿El script de instalación se ha ejecutado sin errores para rpi1?	Sí
¿El script de instalación se ha ejecutado sin errores para rpi2?	Sí
¿El script de instalación se ha ejecutado sin errores para rpi3?	Sí
¿Los componentes del sistema en rpi1 inician automáticamente tras reiniciar el equipo?	Sí
¿Los componentes del sistema en rpi2 inician automáticamente tras reiniciar el equipo?	Sí
¿Los componentes del sistema en rpi3 inician automáticamente tras reiniciar el equipo?	Sí
¿El agente de Zabbix en rpi1 es capaz de comunicarse con Zabbix en ultraheroe?	Sí
¿Los scripts de rpi1 son capaces de comunicarse con rpi2_api?	Sí
¿Rpi2_api es capaz de comunicarse con el Philips® Hue bridge?	Sí
¿Rpi3_api es capaz de comunicarse con rpi2_api?	Sí
¿Rpi3_api es capaz de comunicarse con instalador?	Sí
¿Rpi3_api es capaz de comunicarse con web?	Sí
¿El dashboard (gatsby) de rpi3 es capaz de comunicarse con rpi2_api?	Sí

Tabla 88. Comprobaciones prueba de despliegue completa.

Como se puede observar en la tabla 88 que define las tareas a comprobar durante la **prueba de despliegue completa**, este proceso se ha completado **sin contratiempos**, cumpliendo con todas las comprobaciones previstas en el mismo.

En las sección posterior, Evaluación, se estudian en más detalle las pruebas realizadas al sistema.

Plan de formación

Plan de formación a usuarios

Dado que el sistema es mayormente pasivo, es decir, **no requiere interacción con el usuario**, se ha definido un plan de formación a usuarios bastante simple. Asimismo y por el mismo motivo, no se dispone de un manual de usuario, pues se ha considerado innecesario.

Durante la semana establecida en la planificación del proyecto para formación de los usuarios, se ha mantenido una **reunión de presentación del producto desarrollado** con la totalidad de los técnicos del LDI. En la misma, se ha hecho un repaso de las funcionalidades del sistema, dedicando especial atención a explicar los aspectos de la interfaz del mismo.

Se ha explicado la disposición de los elementos de la interfaz, su funcionamiento (origen de los datos, período de actualización, etc.) y significado. Asimismo, se ha explicado las situaciones de alerta que identifica el sistema así como las alarmas que el mismo ejecuta en estas situaciones. Por último, se ha indicado cómo acceder a la interfaz del sistema desde los equipos de trabajo de los técnicos.

Este proceso ha sido más a modo de recordatorio, ya que la metodología de desarrollo *agile* y la entrega continua del producto han permitido en todo momento que los técnicos interactuaran con el sistema durante la implementación del mismo. La respuesta de los técnicos al **proceso de formación se ha valorado positivamente**, pues una vez finalizado este proceso son capaces de comprender el sistema sin ayuda externa.

Plan de formación de mantenimiento

Debido al uso de tecnologías libres, todo el código fuente y componentes del sistema son **accesibles y modificables por los técnicos del LDI**. Asimismo, el diseño de componentes del sistema, que definen una arquitectura modular, facilita la **implementación de funcionalidades** posteriores y la **modificación del sistema** si las condiciones de servicio en el LDI se vieran modificadas en el futuro.

Como se ha comentado en el capítulo de Implementación, todos los componentes software necesarios para el funcionamiento del sistema, así como los componentes software adicionales (como el mencionado script de instalación `install.sh`), se encuentran en un **repositorio git** en un servidor de la infraestructura del LDI. Estos componentes, eliminando claves y tokens serán liberados y el repositorio duplicado y accesible públicamente desde GitHub [28] una vez sea publicada esta memoria en la biblioteca de la universidad.

Este repositorio se encuentra **completamente documentado** en ficheros Markdown en los que se explica el contenido y arquitectura de cada fichero de código, así como su identificación y funcionalidad dentro de los componentes del sistema. Cada fichero de código por su parte se encuentra dividido en funciones y estas, así como los bloques de código que forman el cuerpo de las mismas, se encuentran completamente **comentados y relacionados con las explicaciones realizadas en los ficheros Markdown** mencionados anteriormente. La disposición de ficheros y carpetas es idéntica a la definida en las secciones del capítulo Diseño.

Tras completar esta documentación del repositorio, se ha presentado la misma a los **técnicos del LDI nuevamente reunidos**. Con estos se ha puesto de acuerdo diferentes aspectos de la documentación para completarla y asegurar que la misma es **fiel al sistema desarrollado**. Asimismo, se ha discutido con los técnicos la implementación final ya desplegada, donde se han planteado **hipotéticas situaciones futuras** que requerirían de modificar el sistema. Esto ha provocado la inclusión en la documentación del repositorio de un apartado acerca de **cómo modificar el sistema ante cambios en la infraestructura o el entorno**.

Evaluación

Plan de pruebas

Para asegurar la calidad del sistema entregado se definen dos planes de pruebas. Un plan de **pruebas unitarias** que aseguran la correcta funcionalidad de los componentes del sistema, y un plan de **pruebas de integración** que aseguran que los distintos componentes del sistema se comunican y coordinan correctamente entre sí.

Si bien **la mayoría de errores han sido corregidos durante la fase de implementación**, ya que la entrega continua propicia la corrección de errores durante dicha fase, los planes de pruebas definidos a continuación proporcionan una **capa extra de calidad** al sistema y han sido posibles gracias a una planificación previsora. Estos planes de pruebas se han definido y llevados a cabo tras el desarrollo del sistema.

Se presentan únicamente las pruebas que se han realizado **posteriormente** a la fase de implementación, obviando las que se han realizado durante dicha fase como parte de la entrega continua. Las distintas pruebas están definidas por una tabla que cumple la la siguiente plantilla:

Identificador	
Título	
Componente	
Precondiciones	
Postcondiciones	
Entrada	
Salida	

Tabla 89. Plantilla pruebas.

Cada uno de los campos de la tabla 89 tiene el siguiente significado:

- **Identificador:** código que identifica unívocamente a cada prueba. Sigue la forma *PU-XX* o *PI-XX*, donde *PU* y *PI* son las siglas para *Prueba Unitaria* y *Prueba de Integración* respectivamente, y *XX* representa el identificador numérico de la prueba (01-99).
- **Título:** descripción breve de la prueba.
- **Componente:** componente o componentes que son objeto de la prueba.
- **Precondiciones:** condiciones previas que deben cumplirse para proceder con la prueba.
- **Postcondiciones:** estado esperado del sistema tras completarse la prueba.
- **Entrada:** entrada utilizada en la prueba.
- **Salida:** salida obtenida tras la prueba.

Los planes de pruebas definidos a continuación han sido **completados con éxito**.

Pruebas unitarias

Se ha definido un conjunto de pruebas unitarias para añadir una capa extra de calidad al testeo realizado durante el desarrollo. A continuación se muestran **algunas** de las pruebas realizadas.

Por supuesto, **el sistema ha sido sometido a una batería de pruebas más extensa de la que se muestra aquí** con el fin de asegurar la calidad del software implementado. No se puede obviar que **la fase de pruebas es una de las fases más importantes** en cualquier proyecto de ingeniería. Sin embargo, el número de pruebas es muy elevado y muchas pruebas son similares por lo que se ha decidido no incluirlas todas en la memoria para no sobrecargar esta, incluyendo únicamente un subconjunto representativo a modo de ilustración. El conjunto completo de pruebas se obtiene con variaciones de la pruebas que conforman este subconjunto.

Además de las pruebas mostradas a continuación, se ha probado el sistema contra las diversas configuraciones de peticiones erróneas o malformadas, inyecciones, situaciones anómalas o de alerta, cambios imprevistos en el entorno y situaciones no concebidas en el normal funcionamiento del sistema.

Las pruebas a los endpoints de las API se han realizado con la herramienta Postman [40].

PU-01	
Título	Obtención del estado del CPD (rpi2_api).
Componente	Rpi2_api (rpi2)
Precondiciones	El subsistema del CPD debe estar en funcionamiento.
Postcondiciones	El estado del sistema no se ve alterado. Se devuelve un código HTTP 200 y un JSON con todos los datos del subsistema.
Entrada	Petición GET a `/cpd-status`
Salida	Código HTTP 200 y fichero JSON con los datos de temperatura, humedad y SAI (UPS) del CPD: `{ "temperature": 20.5, "humidity": 32.4, "ups status (LDI rack)": "online" }`

Tabla 90. PU-01: Obtención del estado del CPD (rpi2_api).

PU-02	
Título	Obtención de la temperatura del CPD mediante parámetros en la URL (rpi2_api).
Componente	Rpi2_api (rpi2)
Precondiciones	El subsistema del CPD debe estar en funcionamiento.
Postcondiciones	El estado del sistema no se ve alterado. Se devuelve un código HTTP 200 y un JSON con el valor de temperatura del CPD.
Entrada	Petición GET a `/cpd-status?temp`
Salida	Código HTTP 200 y fichero JSON con el formato: `{"temperature": 20.4}`

Tabla 91. PU-02: Obtención de la temperatura del CPD mediante parámetros en la URL (rpi2_api).

PU-03	
Título	Obtención de la temperatura y humedad del CPD filtrando más de un parámetro en la URL (rpi2_api).
Componente	Rpi2_api (rpi2)
Precondiciones	El subsistema del CPD debe estar en funcionamiento.
Postcondiciones	El estado del sistema no se ve alterado. Se devuelve un código HTTP 200 y un JSON con los valores de temperatura y humedad.
Entrada	Petición GET a `/cpd-status?temp&hum`
Salida	Código HTTP 200 y fichero JSON con el formato: `{"temperature": 20.5, "humidity": 32.3}`

Tabla 92. PU-03: Obtención de la temperatura y humedad del CPD filtrando dos parámetros en la URL (rpi2_api).

PU-04	
Título	Actualización de la temperatura del CPD (rpi2_api).
Componente	Rpi2_api (rpi2)
Precondiciones	El subsistema del CPD debe estar en funcionamiento.
Postcondiciones	El valor de la temperatura se ve actualizado en el sistema. Se devuelve un código HTTP 200 y un JSON con un mensaje de aceptación.
Entrada	Petición POST autenticada con Bearer token a `/cpd-status` y cuerpo JSON: `{"Temp": 0.0}`
Salida	Código HTTP 200 y fichero JSON con el formato: `{"message": "OK"}`. Una petición GET posterior muestra el valor "0.0" para la temperatura.

Tabla 93. PU-04: Actualización de la temperatura del CPD (rpi2_api).

PU-05	
Título	Petición con método no permitido (rpi2_api).
Componente	Rpi2_api (rpi2)
Precondiciones	El subsistema del CPD debe estar en funcionamiento.
Postcondiciones	El estado del sistema no se ve alterado. Se devuelve un código HTTP 405 y un JSON con un mensaje de error.
Entrada	Petición PUT a `/cpd-status` con cuerpo cualquiera.
Salida	Código HTTP 405 y fichero JSON con el formato: `{"error": "Method Not Allowed"}`.

Tabla 94. PU-05: Petición con método no permitido (rpi2_api).

PU-06	
Título	Petición POST sin cabecera de autenticación (rpi2_api).
Componente	Rpi2_api (rpi2)
Precondiciones	El subsistema del CPD debe estar en funcionamiento.
Postcondiciones	El estado del sistema no se ve alterado. Se devuelve un código HTTP 401 y un JSON con un mensaje de error.
Entrada	Petición POST sin cabecera de autenticación a `/cpd-status` y cuerpo cualquiera.
Salida	Código HTTP 401 y fichero JSON con el formato: `{"error": "Authorization header not provided or empty"}`.

Tabla 95. PU-06: Petición POST sin cabecera de autenticación (rpi2_api).

PU-07	
Título	Petición POST con cabecera de autenticación vacía (rpi2_api).
Componente	Rpi2_api (rpi2)
Precondiciones	El subsistema del CPD debe estar en funcionamiento.
Postcondiciones	El estado del sistema no se ve alterado. Se devuelve un código HTTP 401 y un JSON con un mensaje de error.
Entrada	Petición POST con cabecera de autenticación vacía a `/cpd-status` y cuerpo cualquiera.
Salida	Código HTTP 401 y fichero JSON con el formato: `{"error": "Authorization header not provided or empty"}`.

Tabla 96. PU-07: Petición POST con cabecera de autenticación vacía (rpi2_api).

PU-08	
Título	Petición POST con cabecera de autenticación con token erróneo (rpi2_api).
Componente	Rpi2_api (rpi2)
Precondiciones	El subsistema del CPD debe estar en funcionamiento.
Postcondiciones	El estado del sistema no se ve alterado. Se devuelve un código HTTP 401 y un JSON con un mensaje de error.
Entrada	Petición POST con cabecera de autenticación con token erróneo a `/cpd-status` y cuerpo cualquiera.
Salida	Código HTTP 401 y fichero JSON con el formato: `{"error": "Unauthorized"}`.

Tabla 97. PU-08: Petición POST con cabecera de autenticación con token erróneo (rpi2_api).

PU-09	
Título	Mal formato del cuerpo de la petición, llave de cierre (rpi2_api).
Componente	Rpi2_api (rpi2)
Precondiciones	El subsistema del CPD debe estar en funcionamiento.
Postcondiciones	El estado del sistema no se ve alterado. Se devuelve un código HTTP 400 y un JSON con un mensaje de error.
Entrada	Petición POST autenticada con Bearer token a `/cpd-status` y cuerpo JSON: `{"Temp": 0.0`.
Salida	Código HTTP 400 y fichero JSON con el formato: `{"error": "Bad Request"}`.

Tabla 98. PU-09: Mal formato del cuerpo de la petición, llave de cierre (rpi2_api).

PU-10	
Título	Simulacro de alarma temperatura elevada en el CPD.
Componente	Rpi2_api (rpi2)
Precondiciones	Rpi2_api se encuentra en ejecución y controlando la situación en el CPD con los datos que se le envían. Se envía un dato de temperatura falso >30.0.
Postcondiciones	La bombilla Wi-Fi empieza a parpadear en rojo. Suena la alarma sonora en los altavoces del monitor conectado a rpi2.
Entrada	Petición POST autenticada con Bearer token a `/cpd-status` y cuerpo JSON: `{"Temp": 30.1}`.
Salida	Código HTTP 200 y fichero JSON con el formato: `{"message": "OK"}`. La bombilla Wi-Fi empieza a parpadear en rojo al instante. Suena la alarma sonora en los altavoces del monitor conectado a rpi2.

Tabla 99. PU-10: Simulacro de alarma temperatura elevada en el CPD.

Pruebas de integración

Se han definido un conjunto de pruebas de integración para asegurar y verificar el funcionamiento coordinado y cohesionado de los componentes. A continuación se muestran **algunas** de las pruebas realizadas.

Al igual que ocurre para las pruebas unitarias, **el sistema ha sido sometido a una batería de pruebas más extensa de la que se muestra aquí**. Sin embargo, por los mismos motivos anteriores relacionados con su cardinalidad y similitud, se ha decidido no incluir todas las pruebas en la memoria, incluyendo únicamente un subconjunto representativo a modo de ilustración.

Además de las pruebas mostradas a continuación, se ha probado el sistema contra toda clase de interacciones entre sus elementos, componentes y subsistemas; con el objetivo de asegurar y verificar un funcionamiento global del sistema coordinado y cohesionado entre sus componentes.

PI-01	
Título	Arranque automático del servicio de temperatura del CPD.
Componente	cpd_temperature.service (rpi1)
Precondiciones	El subsistema del CPD está funcionando. Se reinicia rpi1.
Postcondiciones	El servicio debe arrancar al inicio y comenzar a monitorizar la temperatura del CPD, depositando el valor leído en `/tmp/last_temp.txt`.
Entrada	Ninguna.
Salida	Tras reiniciar rpi1, el comando `systemctl status cpd_temperature.service` indica que el servicio ha arrancado con normalidad. El último valor leído se encuentra en `/tmp/last_temp.txt`.

Tabla 100. PI-01: Arranque automático del servicio de temperatura del CPD.

PI-02	
Título	Monitorización de la iluminación del CPD. Comunicación entre componentes.
Componente	light_watcher.py (rpi1), rpi2_api (rpi2), Philips® Hue bridge (rpi2_api).
Precondiciones	Los componentes del subsistema del CPD están funcionado. La bombilla Wi-Fi se encuentra apagada. Se enciende la luz dentro del CPD.
Postcondiciones	La cámara (rpi1) toma una foto en los próximos 10 segundos y detecta la luz encendida en el CPD. Rpi1 manda esta información a rpi2_api. Rpi2_api solicita encender la bombilla Wi-Fi con una petición al Philips® Hue bridge.
Entrada	Se enciende la luz del CPD. La cámara (rpi1) toma una foto.
Salida	La bombilla Wi-Fi se enciende en menos de 10 segundos.

Tabla 101. PI-02: Monitorización de la iluminación del CPD. Comunicación entre componentes.

PI-03	
Título	Simulacro de alarma corte eléctrico en el rack del LDI.
Componente	SAI, ultraheroe, rpi2_api (rpi2), Philips® Hue bridge (rpi2_api).
Precondiciones	Rpi2_api se encuentra en ejecución y controlando la situación en el CPD con los datos que se le envían. Se suspende en un entorno controlado la conexión eléctrica del rack del LDI
Postcondiciones	Ultraheroe detecta el cambio de estado en el SAI del rack del LDI e informa a rpi2_api. La bombilla Wi-Fi empieza a parpadear en rojo. Suena la alarma sonora en los altavoces del monitor conectado a rpi2.
Entrada	Se acciona el diferencial eléctrico asignado al rack del LDI para cortar la alimentación y forzar el SAI a entrar en modo batería.
Salida	La bombilla Wi-Fi empieza a parpadear en rojo al instante. Suena la alarma sonora en los altavoces del monitor conectado a rpi2.

Tabla 102. PI-03: Simulacro de alarma corte eléctrico en el rack del LDI.

PI-04	
Título	Carga externa del dashboard GatsbyJS.
Componente	web_server.go (rpi3), iptables (rpi3)
Precondiciones	El subsistema de aulas se encuentra en ejecución. Se intenta acceder a la GUI del subsistema desde un navegador web en uno de los equipos de los técnicos.
Postcondiciones	El dashboard de GatsbyJS se carga sin problemas en el navegador del equipo, mostrando los datos del subsistema.
Entrada	Se solicita el dashboard de GatsbyJS desde un navegador chromium y un navegador mozilla firefox desde el equipo del técnico.
Salida	El dashboard de GatsbyJS se carga sin problemas en los dos navegadores del equipo, mostrando los datos del subsistema.

Tabla 103. PI-04: Carga externa del dashboard GatsbyJS.

PI-05	
Título	Reflejo de las actualizaciones de reservas del sistema externo.
Componente	web, rpi3_api (rpi3), dashboard GatsbyJS (rpi3)
Precondiciones	El subsistema de aulas se encuentra en ejecución. Se añade una nueva reserva para el día actual en el sistema externo que el LDI usa para gestionar las reservas. Este sistema actualiza las reservas mostradas en la web del LDI.
Postcondiciones	En los próximos 60 segundos el subsistema de aulas debe detectar el cambio en el sistema de gestión de reservas del LDI y reflejar la nueva reserva en el dashboard de GatsbyJS.
Entrada	Se añade una reserva falsa con datos de prueba al sistema de gestión de reservas del LDI, que es externo e independiente del sistema desarrollado.
Salida	En los próximos 60 segundos el dashboard de GatsbyJS recupera los datos de la reserva desde la web del LDI mediante una petición a rpi3_api y refleja la nueva reserva en su GUI.

Tabla 104. PI-05: Reflejo de las actualizaciones de reservas del sistema externo.

PI-06	
Título	Solicitud a instalador del estado de los equipos.
Componente	instalador, rpi3_api (rpi3), dashboard GatsbyJS (rpi3)
Precondiciones	El subsistema de aulas se encuentra en ejecución. El dashboard de GatsbyJS está mostrando la vista de aulas para el aula F16.
Postcondiciones	En los registros de instalador aparece una conexión SSH desde la dirección IP de rpi3 y en la lista de procesos del sistema aparece en ejecución el script `comprobar_ocupacion.py` por el usuario root desde terminal remota.
Entrada	GatsbyJS muestra la vista del aula F16 y solicita la información a rpi3_api, que ejecuta el comando `comprobar_ocupacion.py` por SSH en instalador.
Salida	El script imprime el estado del aula F16 por salida estándar. Rpi3_api parsea esta información y la devuelve en formato JSON al dashboard de GatsbyJS. El dashboard de GatsbyJS muestra el estado actual de los equipos del aula F16. Se comprueba manualmente que este estado es cierto.

Tabla 105. PI-06: Solicitud a instalador del estado de los equipos.

Estudio de la satisfacción de los usuarios

Tras varias semanas de estar el sistema en funcionamiento se realizó una pequeña **entrevista** con los técnicos del LDI para comprobar la **satisfacción** de los usuarios con el sistema desarrollado.

Los técnicos se han mostrado satisfechos con el sistema desarrollado. El sistema cumple con todos los requisitos iniciales de usuario. Además, su implantación ha ahorrado tiempo a los técnicos durante las tareas diarias a la vez que les ha ayudado a planificar mejor algunas tareas de mantenimiento. Asimismo, **el sistema ha permitido detectar situaciones que se han de corregir y que se ignoraban hasta la fecha**, las cuales son detalladas en el capítulo Conclusiones en su sección Relativas al sistema desarrollado.

Conclusiones y trabajos futuros

Conclusiones

Relativas al sistema desarrollado

Como se menciona en el apartado anterior de Estudio de la satisfacción de los usuarios, **el cliente se ha mostrado satisfecho con el sistema** y este además ha permitido detectar las siguientes situaciones hasta ahora desconocidas:

- Todos los lunes por la mañana, entre las 10:00 y las 11:00, el rack del LDI pierde alimentación eléctrica y el SAI entra en modo batería. Esto es debido a un **micro corte eléctrico** de apenas 1-2 segundos. Los técnicos lo están investigando, pero se cree que el SAI se desconecta voluntariamente de la red para realizar una prueba semanal de la batería y las estimaciones de carga.
- El sistema de **climatización del edificio Torres Quevedo afecta** notoriamente a la temperatura dentro del CPD. Parece ser que hay alguna fuga en la instalación, pues el sistema de climatización del edificio teóricamente está desactivado para la sala del CPD (4.0.F09). Se ha dado la incidencia oportuna a través de las canales habilitados para ello y el personal de mantenimiento de la universidad se encuentra actualmente revisando la instalación.

Figura 23: El sistema revela como la climatización del ed. Torres Quevedo afecta a la climatización del CPD.

En la figura 23 se muestra un ejemplo de los datos recogidos durante el mes de febrero, donde se aprecia como **la calefacción del edificio afecta a la temperatura en el CPD**, a pesar de estar esta sala aislada y climatizada de forma separada del resto del edificio mediante sus propias máquinas de aire acondicionado. En la figura se muestran los datos recogidos para la semana del lunes 11 al viernes 15 de febrero de 2019. Los elevaciones de temperatura en la gráfica se corresponden con el horario de actividad de la universidad.

Asimismo, **el sistema implementado satisface todos los objetivos definidos** al inicio de este documento en el apartado de Objetivos y alcance del proyecto. Estos eran:

- Monitorizar el estado del CPD y los servidores: temperatura, humedad, iluminación interior (encendida/apagada) y estado de la red eléctrica.
- Monitorizar el estado de las aulas y los equipos en las mismas: reservas de aulas, ocupación de las aulas y los equipos, y estado de los equipos.
- Presentar toda la información de forma que sea asimilable y comprensible de un vistazo.

- Alertar a los técnicos de situaciones extrañas, anómalas o malfuncionamientos bien en el CPD o en las aulas tan pronto como se detecten.

Por lo motivos expuestos en los párrafos anteriores, se concluye que **el sistema desarrollado cumple con el propósito para el que fue pensado**, suponiendo el producto de este proyecto una herramienta cuya adquisición resulta **muy favorable para el entorno y trabajo diario del LDI**.

En el APÉNDICE C. Galería del sistema se incluye un conjunto de imágenes que comprenden capturas de la interfaz y fotografías del sistema desarrollado.

Relativas al proceso de desarrollo

El proceso de desarrollo del proyecto ha **cumplido con los plazos establecidos en la planificación** del mismo, pues las distintas fases han comenzado y finalizado en el tiempo programado. Asimismo, la utilización de una metodología *agile* con entrega continua durante la implementación ha propiciado la adaptación y evolución del producto para **suplir correctamente las necesidades y objetivos del cliente**.

Con lo cual, se concluye que **el proceso de desarrollo ha sido positivo** al completarse este sin incidentes remarcables. Asimismo, se considera un **acierto la decisión de emplear una metodología *agile*** como Kanban, pues al llevarse a cabo el desarrollo del producto en el mismo despacho donde se iba implantar y junto al cliente que lo iba a utilizar esta metodología ha permitido crear un **producto ajustado y personalizado** a las necesidades del cliente y su infraestructura.

Personales. Valor y aporte académico

Los **conocimientos** que he **adquirido** durante mis estudios del grado en ingeniería informática han permitido la realización de este proyecto, del cual estoy especialmente orgulloso por el buen recibimiento de mis compañeros en el LDI y por poder comprobar personalmente, durante mis últimos días en las prácticas extracurriculares, la utilidad del mismo y la medida en la que simplifica el trabajo diario.

La realización de este proyecto no hubiese sido posible, como digo, sin los conocimientos y competencias que he adquirido en las asignaturas del grado. Especialmente:

- **Programación.** Asignatura en la que aprendí a codificar y entender código y sin la que no podría haber adquirido posteriormente por mi cuenta los conocimientos de Python, Go y JavaScript que han requerido el sistema, ni entender el funcionamiento de los frameworks de este último.
- **Redes de ordenadores.** El conocimiento de redes de computadores y la pila de protocolos, en especial las capas de Internet y aplicación, han permitido desarrollar un canal de comunicación eficaz y eficiente entre los distintos componentes del sistema, a la vez que ha

propiciado el aseguramiento de este canal contra agentes externos.

- **Ingeniería del software**. Esta asignatura me mostró la importancia de la creación de un código limpio y mantenible, así como el de la ordenación del mismo en funciones y ficheros separados y bien diferenciados en su funcionalidad y significado. Estos principios los he llevado como estandarte a la hora de implementar el sistema.
- **Dirección de proyectos de desarrollo de software.** Sin cuyos conocimientos y experiencia hubiese sido incapaz de planificar correctamente el ciclo de vida del sistema. Asimismo, las competencias adquiridas en esta asignatura me han permitido aprender y aplicar por mi cuenta una metodología de desarrollo *agile* y generar toda la documentación del proceso y el producto que se aprecia en esta memoria, si bien reducida a la carga de 12 ECTS.
- **Interfaces de usuario.** Donde he aprendido la importancia de las buenas prácticas, heurísticas de diseño, y el proceso cognitivo del usuario para diseñar interfaces de usuario simples e intuitivas, uno de los objetivos del sistema.
- **Sistema operativos.** A esta asignatura le debo parte de mis conocimientos de bash y la arquitectura de GNU/Linux, necesarios todos ellos para implantar y desplegar el sistema, así como trabajar a bajo nivel con los elementos hardware de las Raspberry Pi.
- **Criptografía y seguridad informática.** En esta asignatura adquirí los conocimientos de autenticación y firma necesarios para asegurar la comunicación entre los distintos componentes del sistema, así como la utilización de sistemas criptográficos de clave pública (como RSA) utilizados en el proyecto.

Personalmente puedo concluir que **he adquirido los conocimientos y competencias** a los que aspiraba cuando cuatro años atrás solicité mi admisión al grado en ingeniería informática en nuestra universidad. Estoy orgulloso de haberlo demostrado con la realización de este sistema con el que además dejo un trabajo que **resulta de utilidad** para un servicio como el LDI; el cual es un servicio que, como he podido conocer desde dentro, resulta imprescindible para el correcto funcionamiento y la calidad de mi grado.

Trabajos futuros

A continuación se relatan algunas mejoras y trabajos futuros que se podrían aplicar al sistema, de los cuales muchos tengo **intención de realizarlos yo mismo** durante el tiempo que permanezca en las prácticas extracurriculares en el LDI. No obstante, me gustaría recordar que el trabajo aquí realizado responde a la asignación horaria del TFG, que se corresponde con 12 ECTS (aproximadamente 360 horas), por lo que no se ha dispuesto del tiempo material para abordar estas tareas adicionales dentro del TFG.

Integración con Google Calendar

En el LDI se utiliza la herramienta **Google Calendar** [41] con las cuentas de la universidad para programar y organizar algunas tareas de mantenimiento, reuniones, u otras actividades que re-

quiera la coincidencia horaria de varios técnicos y/o becarios. El subsistema de aulas en un buen candidato para, en el panel principal, mostrar una pantalla con las **tareas pendientes para ese día** o quizás incluso esa semana.

Arquitectura distribuida utilizando Kubernetes

Actualmente el sistema reparte sus componentes principalmente entre tres minicomputadores Raspberry Pi. Esto supone que el sistema tiene una menor fiabilidad en cuanto a errores de hardware que si se ejecutara sobre un único equipo, ya que el malfuncionamiento de uno solo de estos puede afectar al resto del sistema. Es decir, hay más **puntos potenciales de fallo**.

Por la arquitectura y envergadura del sistema, sería asumible trasladar la operativa del mismo desde directamente sobre el Raspbian de las raspberrys hacia nodos de **Kubernetes** [42]. La utilización de Kubernetes, cuya aplicación a este proyecto sería bastante directa y natural, proporcionaría una **capa adicional de compatibilidad** al sistema al estar todo el entorno necesario dentro de contenedores de Docker [43], pudiendo este sistema replicarse y lanzarse en cualquier otra infraestructura hardware, no necesariamente Raspberrys Pi.

Además, la utilización de Kubernetes dotaría al sistema de **mayor resistencia ante fallos**, ya que si un nodo se para por cualquier motivo, podría relanzarse en otro hardware. De la misma forma, esto permitiría realizar un mantenimiento a fondo o reinstalación de los minicomputadores sin necesidad de suspender temporalmente la operatividad del sistema.

Inteligencia artificial para adelantar acontecimientos en el CPD

El control del entorno del CPD es una tarea muy importante en el LDI. Con la inclusión de técnicas de inteligencia artificial, y en especial de **aprendizaje automático**, el sistema podría aprender a reconocer ciertas situaciones repetitivas o comunes en el CPD; así como situaciones simuladas con las que se podría entrenar al sistema. Con este conocimiento, se podría **predecir e informar a los técnicos** del LDI de situaciones en el CPD que deben antender **antes** incluso de que estas situaciones comiencen.

Inteligencia artificial para programar mantenimiento en las aulas

Los técnicos deben asegurar el correcto funcionamiento de los equipos y ecosistemas en las aulas, para lo que realizan regularmente labores de mantenimiento, pero que no deben interferir con el normal flujo de trabajo y utilización de las aulas. Con la inclusión de técnicas de **inteligencia artificial** el sistema podría **recomendar** qué períodos son los más idóneos para el mantenimiento, o qué equipos sufren mayor desgaste y por lo tanto habría que revisar con especial atención. Asimismo podría **predecir**, si se entrena al sistema con datos históricos, cuándo va a fallar un equipo o un componente con cierta probabilidad o en una ventana de tiempo.

Referencias

[1] *Apache HTTP Server Project*, The Apache Software Foundation, ver. 2.4. [En línea]. Disponible en: https://httpd.apache.org/ (Accedido: 05-05-2019).

[2] *Minicomputador Raspberry Pi. Modelo 3B+*, The Raspberry Pi Foundation. [En línea]. Disponible en: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/ (Accedido: 24-04-2019).

[3] *Módulo SenseHat Raspberry Pi*, The Raspberry Pi Foundation. [En línea]. Disponible en: https://www.raspberrypi.org/products/sense-hat/ (Accedido: 24-04-2019).

[4] *Zabbix*, Zabbix LLC, ver. 3.0.7. [En línea]. Disponible en: https://www.zabbix.com/ (Accedido: 24-04-2019).

[5] *Grafana*, Grafana Labs, ver. 6.1.3. [En línea]. Disponible en: https://grafana.com/ (Accedido: 24-04-2019).

[6] *Kibana*, Elasticsearch B.V. Ver. 6.2.0. [En línea]. Disponible en: https://www.elastic.co/es/products/kibana (Accedido: 24-04-2019).

[7] *Keen dashboard*, Keen, ver. estable. [En línea]. Disponible en: https://keen.github.io/dashboards/ (Accedido: 24-04-2019).

[8] *Zabbix plugin for Grafana*, Alexander Zobnin, ver. 3.10.2. [En línea]. Disponible en: https://grafana.com/plugins/alexanderzobnin-zabbix-app (Accedido: 24-04-2019).

[9] P. S. Foundation, «PEP 373 – Python 2.7 Release Schedule», *Python Developer's Guide*, [En línea]. Disponible en: https://www.python.org/dev/peps/pep-0373/ (Accedido: 11-06-2019).

[10] «Ocupación diaria», *Laboratorio del Dpto. de Informática*, [En línea]. Disponible en: https://www.lab.inf.uc3m.es/informacion/ocupacion-de-las-aulas/ocupacion-diaria/ (Accedido: 25-04-2019).

[11] L. W. F. Whitcombe, «Análisis, Diseño e Implementación de Herramientas de Gestión para Laboratorios Docentes del Departamento de Informática de la UC3M», Trabajo fin de grado, Dpto. de Informática, Universidad Carlos III de Madrid, Madrid, España, 2018.

[12] *Angular*, Google LLC, ver. 7.0.0. [En línea]. Disponible en: https://angular.io/ (Accedido: 29-04-2019).

[13] *React*, Facebook Inc, ver. 16.6.0. [En línea]. Disponible en: https://reactjs.org/ (Accedido: 29-04-2019).

[14] *Vue.js*, Evan You, ver. 2.X. [En línea]. Disponible en: https://vuejs.org/ (Accedido: 29-04-2019).

[15] M. Butusov, «ReactJS vs Angular5 vs Vue.js - What to choose in 2018?», *TechMagic*, [En línea]. Disponible en: https://blog.techmagic.co/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018/ (Accedido: 29-04-2019).

[16] *GatsbyJS*, Gatsby Inc, ver. 2. [En línea]. Disponible en: https://www.gatsbyjs.org/ (Accedido: 29-04-2019).

[17] *Next.js*, ZEIT Inc, ver. 7. [En línea]. Disponible en: https://nextjs.org/ (Accedido: 29-04-2019).

[18] *Create React App*, Facebook Inc, ver. 2. [En línea]. Disponible en: https://facebook.github.io/create-react-app/ (Accedido: 29-04-2019).

[19] *REGLAMENTO (UE) 2016/679 DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/46/CE (Reglamento general de protección de datos)*, Boletín Oficial del Estado, 2016. [En línea]. Disponible en: https://www.boe.es/doue/2016/119/L00001-00088.pdf (Accedido: 06-05-2019).

[20] «¿Qué es el software libre?», *El sistema operativo GNU*, [En línea]. Disponible en: https://www.gnu.org/philosophy/free-sw.html.es (Accedido: 06-05-2019).

[21] *MIT License*, Choose an open source license. GitHub Inc. [En línea]. Disponible en: https://choosealicense.com/licenses/mit/ (Accedido: 06-05-2019).

[22] *Apache License 2.0*, Choose an open source license. GitHub Inc. [En línea]. Disponible en: https://choosealicense.com/licenses/apache-2.0/ (Accedido: 06-05-2019).

[23] «History and License», *Python 3 Docs*, [En línea]. Disponible en: https://docs.python.org/3/license.html (Accedido: 06-05-2019).

[24] «LICENSE», *The Go Programming Language*, [En línea]. Disponible en: https://golang.org/LICENSE (Accedido: 06-05-2019).

[25] O. America, «JavaScript», pat. estadounidense 75 026 640. [En línea]. Disponible en: https://tsdr.uspto.gov/#caseNumber=75026640&caseType=SERIAL_NO&searchType=statusSearch (Accedido: 06-05-2019).

[26] *Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia*, Boletín Oficial del Estado, 1996. [En línea]. Disponible en: https://www.boe.es/diario_boe/txt.php?id=BOE-A-1996-8930 (Accedido: 07-05-2019).

[27] *DIRECTIVA 2009/24/CE DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 23 de abril de 2009 sobre la protección jurídica de programas de ordenador*, Boletín Oficial del Estado, 2009. [En línea]. Disponible en: https://www.boe.es/doue/2009/111/L00016-00022.pdf (Accedido: 07-05-2019).

[28] *Sistema de monitorización y alertas de estado para el laboratorio del Departamento de Informática*, Aitor Alonso Núñez (Tairosonloa), 2019. [En línea]. Disponible en: https://github.com/tairosonloa/bachelor_degree_thesis (Accedido: 07-05-2019).

[29] L. Gilibets, «Qué es la metodología Kanban y cómo utilizarla», *Agile y Scrum, iebs*, [En línea]. Disponible en: https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/ (Accedido: 15-05-2019).

[30] *Trello*, Atlassian S.L. [En línea]. Disponible en: https://trello.com/.

[31] «Search salaries and compensation», *Glassdoor Inc*, [En línea]. Disponible en: https://www.glassdoor.com/Salaries/index.htm (Accedido: 02-06-2019).

[32] «Convert my salary to an equivalent hourly wage», *CalcXML*, [En línea]. Disponible en: https://www.calcxml.com/calculators/convert-salary-to-hourly (Accedido: 02-06-2019).

[33] F. Salamanca, «Aire acondicionado y clima urbano», *Revista El Ecologista nº 69*, [En línea]. Disponible en: https://www.ecologistasenaccion.org/?p=12236 (Accedido: 05-05-2019).

[34] *Plan Estratégico 2016 - 2022*, 1.ª ed. Madrid, España: Estugraf Impresores, S.L, 2017. [En línea]. Disponible en: https://hosting01.uc3m.es/semanal3/documents/Plan_estrategico_2016_2022.pdf (Accedido: 05-05-2019).

[35] «Zabbix Documentation 3.0. User parameters», *Zabbix LLC*, [En línea]. Disponible en: https://www.zabbix.com/documentation/3.0/manual/config/items/userparameters (Accedido: 24-04-2019).

[36] *GoHue*, Collinux. [En línea]. Disponible en: https://github.com/collinux/GoHue.

[37] M. Koenig. (17 de mayo de 2009). Industrial Alarm, [En línea]. Disponible en: http://soundbible.com/287-Industrial-Alarm.html (Accedido: 27-05-2019).

[38] *Cloc, Count Lines of Code*, Al Danial, ver. 1.82. [En línea]. Disponible en: https://github.com/AlDanial/cloc (Accedido: 11-06-2019).

[39] *Raspbian Stretch Lite*, Raspberry Pi Foundation, ver. 2019.04.08. [En línea]. Disponible en: https://www.raspberrypi.org/downloads/raspbian/ (Accedido: 07-05-2019).

[40] *Postman*, Postman Inc. [En línea]. Disponible en: https://www.getpostman.com/products (Accedido: 10-06-2019).

[41] *Google Calendar*, Google LLC. [En línea]. Disponible en: https://www.google.com/calendar/about/ (Accedido: 09-06-2019).

[42] «What is Kubernetes», *Kubernetes documentation*, [En línea]. Disponible en: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (Accedido: 09-06-2019).

[43] «Docker overview», *Docker docs*, [En línea]. Disponible en: https://docs.docker.com/engine/docker-overview/ (Accedido: 09-06-2019).

APÉNDICE A. Acrónimos

API	Application Program Interface. Caja negra que ofrece llamadas a funciones.
BSD	Berkeley Software Distribution. Fue un sistema operativo derivado de UNIX.
CPD	Centro de Procesamiento de Datos. Sala donde se encuentran los servidores.
CSS	Cascade Style Sheet (hoja de estilos en cascada).
CU	Caso de Uso.
ECTS	European Credit Transfer and Accumulation System. Medida del trabajo realizado por un estudiante universitario en un asignatura.
FSF	Free Software Foundation. Fundación que vela por la protección y expansión del software bajo licencias libres.
GNU	Sistema operativo de software libre. Parte importante de la FSF y los sistemas Linux.
GPL	GNU General Public License. Licencia de software libre respaldada por GNU y la FSF.
GUI	Graphic User Interface (interfaz gráfica de usuario).
HDMI	High-Definition Multimedia Interface. Cableado que permite la transmisión de imagen y audio.
HTML	HyperText Markup Language. Lenguaje para definir la estructura de sitios web.
HTTP	HyperText Transfer Protocol. Protocolo para tranferencia de hipertexto (contenido web).
IT	Information Technology. Tecnología informática.
LAN	Local Area Network. Red de computadores interconectados fuera de Internet.
LDI	Laboratorio del Departamento de Informática.
MIT	Licencia de software libre permisiva del Instituto de Tecnología de Massachusetts.
PDI	Personal Docente e Investigador.
PI	Prueba de Integración.
PU	Prueba Unitaria.
RD	Real Decreto.
REST	REpresentational State Transfer.
RF	Requisito Funcional.
RGPD	Reglamento General de Protección de Datos.
RNF	Requisito No Funcional.
RU	Requisito de Usuario.

SAI	Sistema de Alimentación Ininterrumpida.
SO	Sistema Operativo.
SSH	Secure SHell. Ejecución de comandos en remoto de forma segura.
TFG	Trabajo Fin de Grado.
UE	Unión Europea.
YAGNI	You Aren't Gonna Need It. En Ingeniería del software, filosofía que argumenta que no se debe agregar nunca una funcionalidad excepto cuando sea necesaria.

APÉNDICE B. Glosario

Bash	Intérprete de comandos de terminal habitual de los sistemas GNU/Linux.
Crawler	Pieza de software encargada de interpretar en contenido de webs diseñadas para seres humanos.
Dashboard	Panel web. Interfaz de usuario destinada para mostrar información de forma gráfica o visualmente entendible.
Demonio	En computación, programa o pieza de software que se ejecuta en segundo plano sin interacción con el usuario. Normalmente se inician automáticamente al inicio del sistema.
Endpoint	Dirección de entrada a un servicio en las arquitecturas orientadas a servicios. Dirección de un recurso de una API REST.
Framework	Conjunto estandarizado de conceptos, prácticas, criterios y herramientas para hacer frente a un problema (el desarrollo de una solución/aplicación).
Git	Software de control de versiones. Un repositorio git permite almacenar el código fuente asociado a un proyecto y su histórico de versiones.
Human readable	Información dispuesta en un formato pensado para su lectura e interpretación por seres humanos, en contraposición a la información formateada para ordenador.
Kanban	Panel que actúa como sistema de tarjetas que controla el flujo de tareas y recursos en un proceso de desarrollo o fabricación.
Loopback	Interfaz de red ficticia que definen los sistemas UNIX like para que distintos componentes o aplicaciones puedas comunicarse mediante protocolos de red sin que el tráfico de la comunicación abandone el sistema.
Markdown	Lenguaje de marcas ligero para dar formato a texto plano con poco esfuerzo.
Rack	Armario destinado al almacenaje de servidores en funcionamiento.
Raspberry Pi	Minicomputador de placa única.
Systemd	Moderno sistema de inicio y gestión de dependencias en el arranque, habitual de las distribuciones GNU/Linux.
UNIX like	Arquitectura de sistema operativo basada en los sistemas UNIX ® de 1969.

APÉNDICE C. Galería del sistema

En el presente apéndice se muestran diversas capturas de pantalla de la interfaz gráfica de usuario del sistema así como algunas fotos del sistema desplegado. A continuación se explican todas ellas.

En la figura 24 se muestra una imagen del minicomputador rpi1 situado dentro del CPD. La imagen muestra cómo se ve la temperatura actual del CPD en el panel led del módulo SenseHat. Asimismo presenta la disposición del módulo de cámara, que cuelga de esta Raspberry y apunta hacia la pared.

La misma figura incluye una imagen con la luz del CPD apagada (izquierda) y con la luz del CPD encendida (derecha).

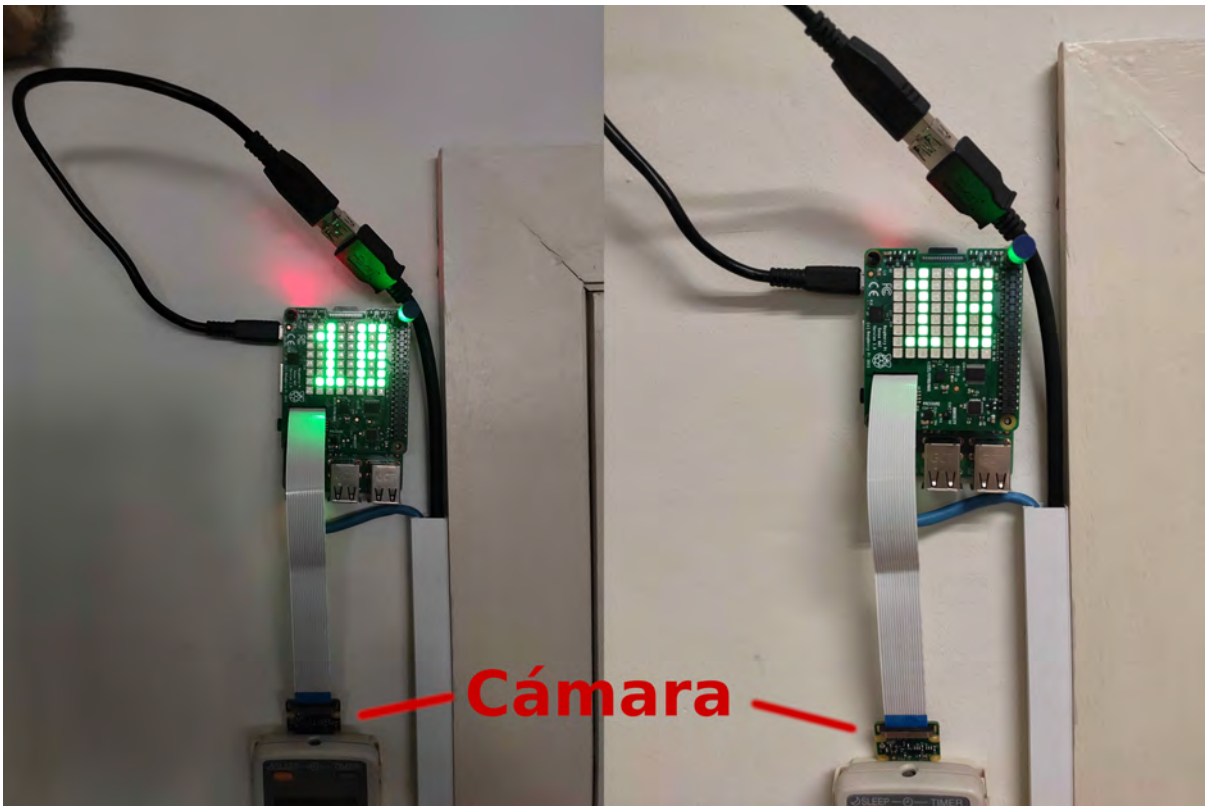

Figura 24: Rpi1 y módulos dentro del CPD. Temperatura actual en el panel led del módulo SenseHat.

La figura 25 muestra los datos y métricas del CPD tal y como se ven en el dashboard de Grafana definido como interfaz gráfica de usuario del subsistema del CPD. Esta es la vista completa, que resume toda la información del CPD, y está pensada para que los técnicos del LDI la consulten desde sus equipos de trabajo. Se corresponde con la definida en el prototipo de diseño (figura 16).

Figura 25: Dashboard de Grafana completo. GUI del subsistema del CPD.

Las figuras 26 y 27 se corresponden con dos fotos del monitor y la bombilla sobre la puerta del CPD, dentro del despacho del LDI. El monitor muestra las vistas de temperatura actual e histórico de temperatura de los últimos 7 días, respectivamente. La bombilla se encuentra encendida indicando que hay alguien haciendo uso del CPD.

Figura 26: Monitor y bombilla de rpi2 sobre la puerta de acceso al CPD. Temperatura actual en el CPD.

Figura 27: Monitor y bombilla de rpi2 sobre la puerta de acceso al CPD. Histórico de temperatura en el CPD.

La figura 28 muestra una captura de la interfaz gráfica de usuario del subsistema de aulas. Se corresponde con la vista de reservas definina en el prototipo de diseño de la figura 17. Contiene los datos de una reserva de prueba para ilustrar la funcionalidad del sistema, puesto que la captura es del mes de junio y ya no hay reservas de aulas porque han acabado las clases.

En la figura se puede apreciar el diseño de la GUI: el panel principal a la izquierda con las tarjetas de reservas, el panel lateral a la derecha con el estado de reserva de las aulas (código de colores) y el número de usuarios por aula, el panel superior con el nombre del servicio, fecha y hora; y el panel inferior con la información resumida del CPD pasando de derecha a izquierda.

Tanto la tarjeta con los datos de la reserva de prueba como el indicador del aula F16 en el panel lateral se encuentran en rojo para indicar que el aula se encuentra actualmente ocupada por la reserva. Como se puede apreciar en la imagen, la reserva es de 9:00 a 11:00 y el reloj de la interfaz muestra las 10:46.

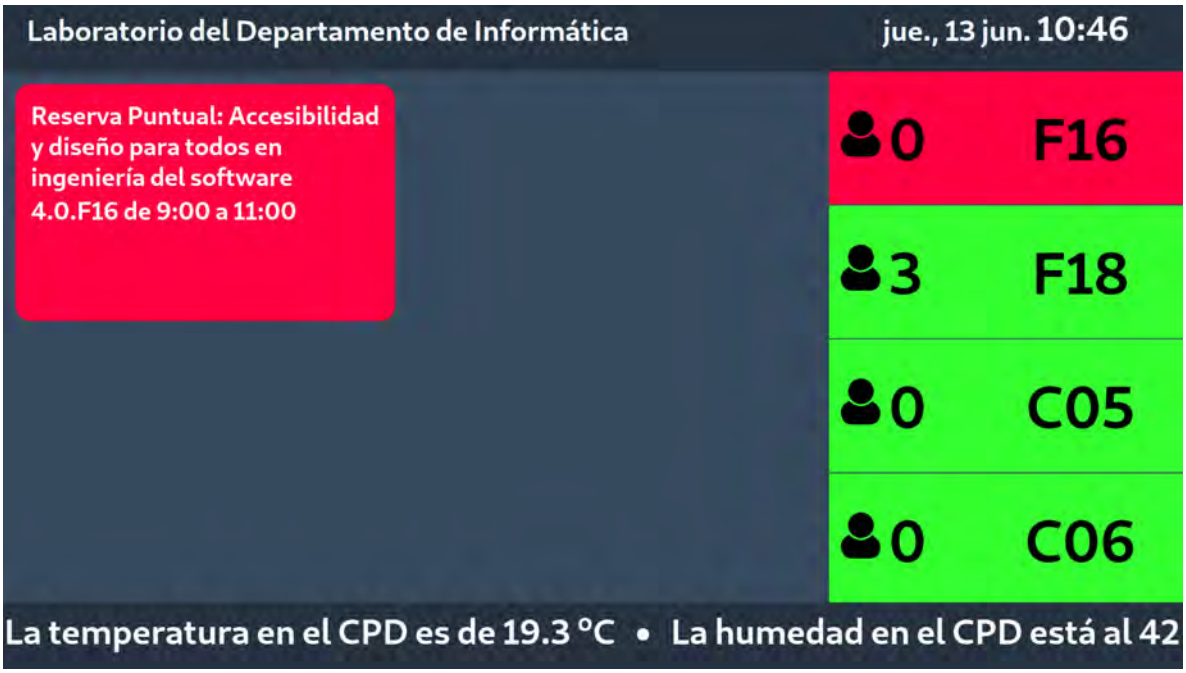

Figura 28: Dashboard de GatsbyJS. Vista de reservas con reserva de prueba. GUI del subsistema del aulas.

La figura 29 muestra la vista de aulas relativa al aula 4.0.F18. En la misma podemos observar que hay tres estudiantes usando el aula, que se han sentado en mesas contiguas, y que uno de ellos está usando el equipo F209 con GNU/Linux y los otros dos los equipos F210 y F212 con Windows.

Figura 29: Dashboard de GatsbyJS. Vista de aulas (4.0.F18). GUI del subsistema del aulas.

Si los equipos estuvieran encendidos pero sin ser usados no aparecería el icono blanco de usuario en los mismos, ni tampoco contaría a efectos del número de usuarios mostrados en el panel lateral.

La figura 30 muestra la vista de aulas relativa al aula 2.2.C05, que tiene una disposición diferente a las aulas del edificio Torres Quevedo. El aula se encuentra cerrada, por lo que están todos los equipos apagados.

Figura 30: Dashboard de GatsbyJS. Vista de aulas (2.2.C05). GUI del subsistema del aulas.

Por último, la figura 31 muestra un aviso de error a la hora de recuperar los datos de las reservas desde la página web. Esto se debe a que la página fue desactivada por unos minutos durante un mantenimiento de la misma.

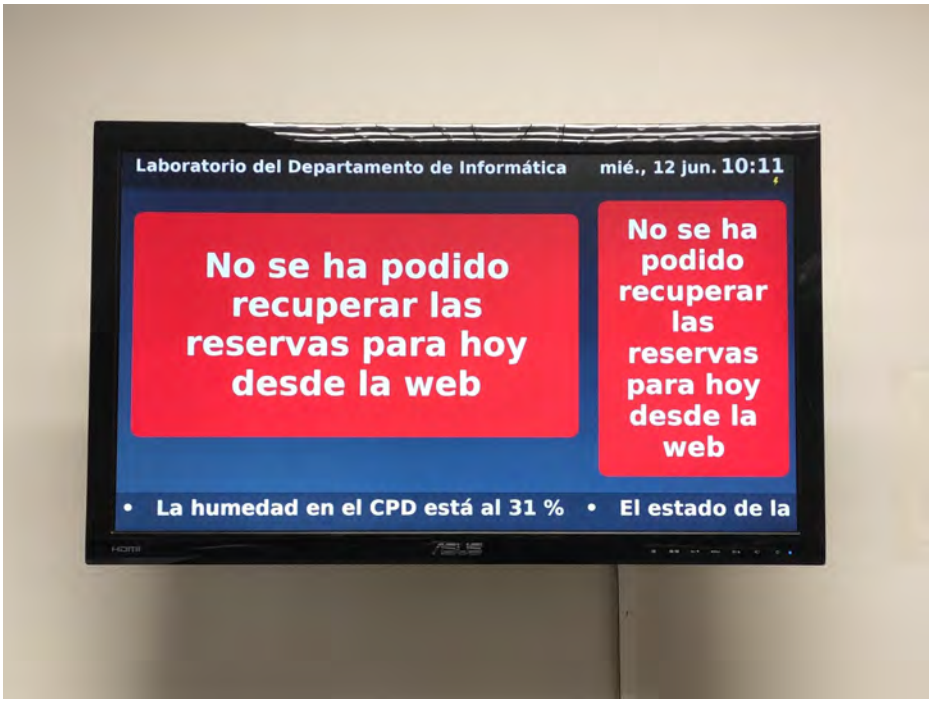

Figura 31: Dashboard de GatsbyJS. Error web caída. Vista de reservas. GUI del subsistema del aulas.